2000

FIR filter design using Jacobian elliptic Sn function with wavelet applications.

(John) Zhengluo. Qiu
University of Windsor

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

FIR FILTER DESIGN USING JACOBIAN ELLIPTIC SN FUNCTION WITH WAVELET APPLICATIONS

by

(John) Zhengluo Qiu

A Thesis
Submitted to the College of Graduate Studies and Research
through Electrical & Computer Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada
2000

ABSTRACT

This thesis presents a new method of digital FIR filter design based on the Jacobian elliptic sine function. In addition to a general design, a special result is obtained for Quadrature Mirror Filters (QMF) for use in multi-resolution filter banks. These are called Quasi-Dyadic FIR filters. Most of the Quasi-Dyadic filter coefficients are $\alpha 2^{-n}$, where α is the scale of the coefficients and n is a positive integer. With these special coefficients, the 'multiply' operation may be changed to a 'shift' operation in assembly coding in signal filtering. Since 'shift' is much faster than the 'multiply' instruction in most microprocessors or DSP chipsets, the Quasi-Dyadic filter is more efficient. For multi-resolution filter banks (wavelet-like), perfect reconstruction (PR) is shown to be possible by a novel strategy. In addition, simpler structures that do not provide PR are shown to be valuable in practical applications. Implementation on a DSP board is also presented.

KEY WORDS: FIR Filter, Elliptic Sn Function, Wavelet, QMF

ACKNOWLEDGMENTS

I thank Dr. James Soltis for helping me understand the excellent idea of using Jacobian functions in FIR design and encouraging me to complete this thesis. Thanks to Dr. Hon K. Kwan for stimulating my interest in digital signal processing technology and making all of this possible.

Special thanks to Dr. Gary W. Rankin, and Dr. Maher A. Sid-Ahmed for spending their precious time to read this thesis and for their comments and suggestions.

Thanks to all of the committee members for being accommodating when I needed it most.

Thanks to Ms. Shirley Ouellette and Mr. Alan Johns for their great advice concerning this document.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

RESEARCH BACKGROUND – LITERATURE REVIEW

1 Introduction

Digital filtering is one of the most important functions in digital signal processing. Digital filter design has been studied since about 1970. It is widely used in many diverse applications, including image processing, speech processing and data communications. The commonly used digital filters are Linear Shift-Invariant (LSI) types. They are easy to analyze, design, and implement.

1.1 Filters Introduction

A filter is like a frequency selective device that shapes the magnitude and/or phase response according to specifications.

Much information exists on the theories, applications, and technologies about design and realization of digital filters. For most applications, it is desirable to design frequency-selective filters. In this case, the desired design specifications are given in the frequency domain by specifying a desired frequency response. One of the essential roles in filter design is designing a high frequency-selective filter with sharp cutoff edges (short transition bands).

However, the ideal sharp edge response has discontinuities in mathematical theories and cannot be realized in practice. Therefore, the filter design consists of finding an implementation filter whose order is low and whose frequency response best approximates the specified ideal magnitude and phase responses which are given as the desired design specifications or constraints. The typical magnitude-frequency terminology of a digital filter is shown in Figure 1-1.

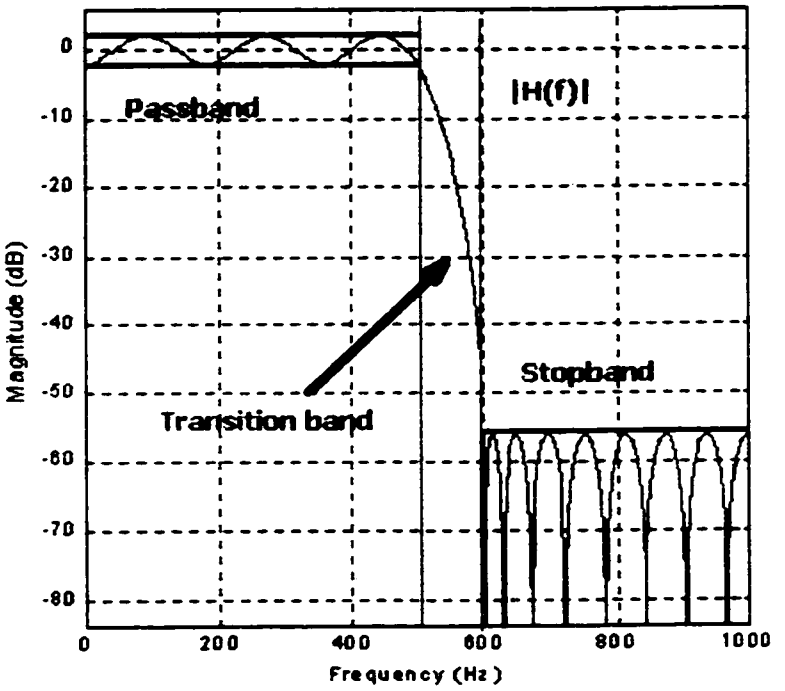


Figure 1.1: Filter terminology

1.1.1 Features of FIR and IIR

There are two main types of LSI: Finite-duration Impulse Response (FIR) filters and Infinite duration Impulse Response (IIR).

FIR filters are those for which the impulse response *h(n)* is non-zero for only a finite number of samples. IIR filters are those for which *h(n)* have an infinite number of non-zero samples. In the FIR case, the samples of the sequence *h(n)* are commonly referred to as the filter coefficients; for the IIR case, the filter coefficients include feedback terms in a difference equation.

Since only the FIR filter is used in this thesis, the following discussion is focused on the FIR case.

1.1.2 General Filter Design Steps

The general design of digital filters is typically done using the following steps:

1) Choose the precise design specifications of the digital filter's desired magnitude and phase responses, filter type (FIR or IIR), filter order, error tolerance, or criteria, based on the application and the detail provided by the user.

2) Approximate the design specifications (of Step 1) of the desired filter type. The requirement is that the obtained filter frequency response best meets the design specifications according to a mathematical error criterion using Chebyshev Approximation, Elliptic Approximation, Bessel Approximation and other mathematical optimization and approximation methods.

3) Realize the filter using the digital technology most suitable for the considered application.

1.2 FIR Filters

FIR filters are always stable and can be designed with a linear phase, although they require more elements in their realization than IIR filters. Their design is typically based on selecting a symmetric impulse response sequence whose length is chosen to meet design specifications. The three most commonly used methods of FIR filter design are window-based methods using the impulse response of ideal filters, frequency sampling and iterative design based on optimal constraints.

FIR filter transfer function has no poles except possibly at *'0'*. That means that they have only zeros at which $H(z)=0$. Therefore, a FIR filter is always stable.

FIR filters can be used in the design of causal linear-phase systems, which are very important and widely used in practice. In fact, in many signal-processing applications, such as speech and image processing, it is desirable to pass some portion of the signal frequency band with minimal distortion. For that purpose, linear-phase systems are particularly desirable since the effect of the linear-phase is a pure time delay.

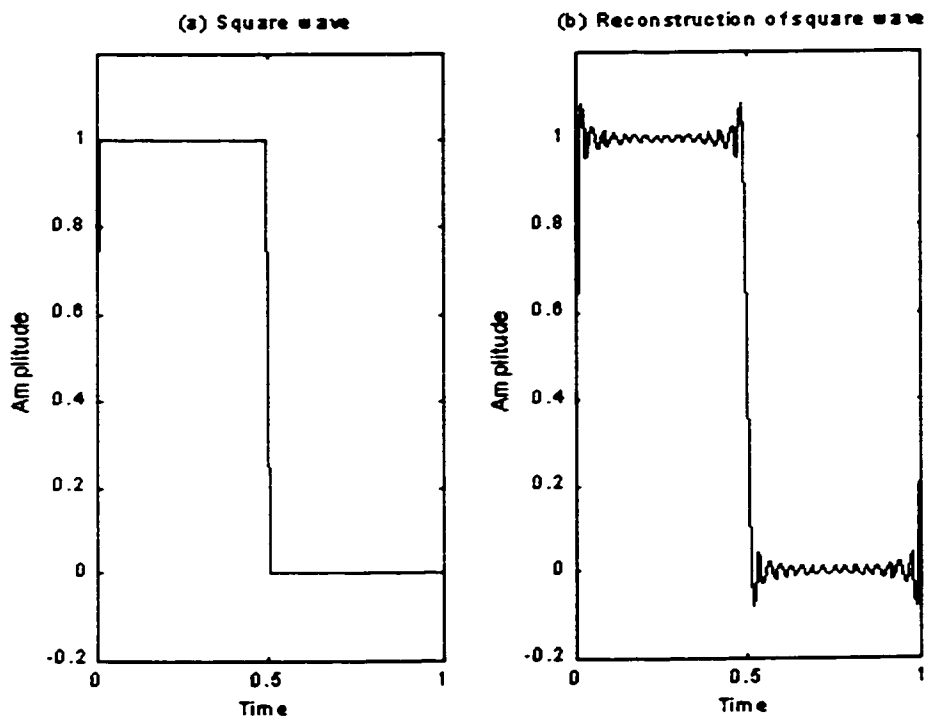


Figure 1.2: Gibbs effect

1.2.1 Gibbs Effect

FIR filter design is based on selecting a symmetric impulse response sequence whose length is chosen to meet design specifications. One of the non-optimal and most straightforward solutions to FIR filter design is the windowing technique. It is clear that the corresponding impulse response samples of an ideal rectangular filter is a Sinc function of infinite length in the time domain.

Square windowing in the time domain reduces the number of samples to a finite order, which has the drawback of the Gibbs phenomena. Namely, this truncation causes the low pass filter frequency response to have ripples in the

passband and stopband. The biggest overshoot is about 9% of the magnitude of the discontinuity (Figure 1.2) [19]. Various windows may be used in the design procedure to reduce this effect. They are Hamming, Hanning, Bartlett, and Kaiser windows.

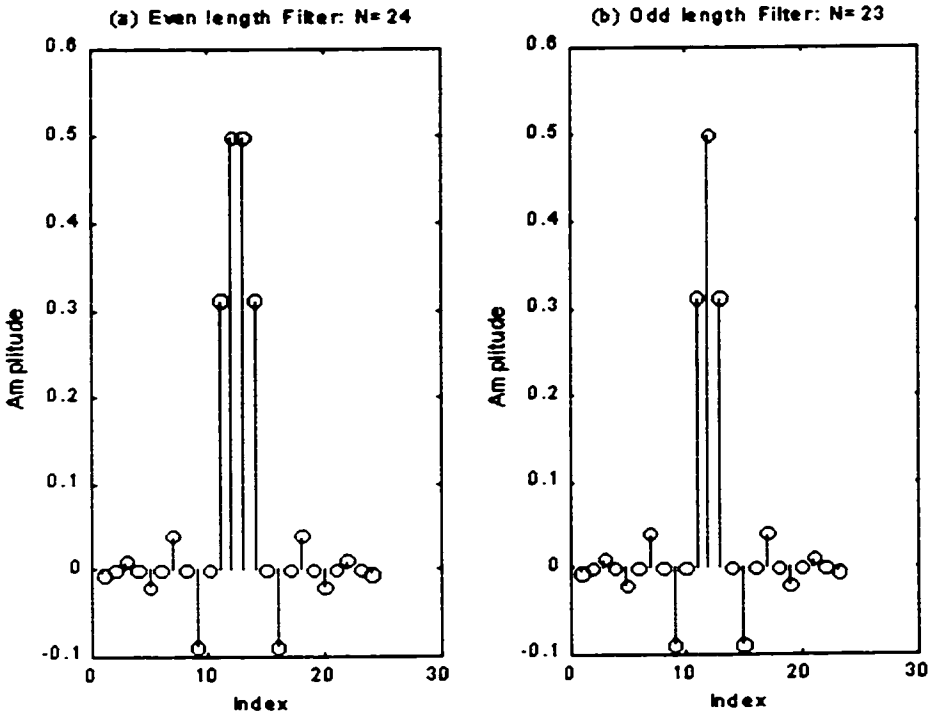


Figure 1.3: Symmetric sequences of filter coefficients

1.2.2 Symmetric Sequences and Linear Phase

An important consideration in filter design is constant delay with no amplitude distortion. That means the filter should possess a linear phase. Symmetric sequences (Figure 1.3) have this character. Nevertheless, symmetric sequences

also cause non-causal system problems. However, such drawbacks can be solved by using suitable delays.

1.2.3 Classification of Linear-Phase Sequences and Applications

There are four types of symmetric sequences. The length N of each filter type can be odd or even. In addition, the center of symmetry may fall on a sample point for odd N or midway between samples for even N. Table 1.1 shows the relationship of the types and their characters and applications.

Table 1.1: Characters and applications of symmetric sequence

	Type 1	Type 2	Type 3	Type4
Length	Odd	Even	Odd	Even
Symmetry	Even	Even	Odd	Odd
Poles	All poles are at z=0 on z-plane			
Zeros	Zeros show conjugate reciprocal symmetry			
Zeros at z=1	Even	Even	Odd	Odd
Zeros at z=-1	Even	Odd	Odd	Even
Applications	LPF,BPF, BSF & HPF	LPF & BPF	BPF, differentiators & Hilbert transformers	BPF, HPF, differentiators & Hilbert transformers

1.2.4 FIR Filters Design Methods

There are three the most commonly used methods for FIR filter design: the window-based method, the frequency sampling method, and the optimal constraint iterative method. The desired FIR filters should represent the impulse response of an ideal filter.

1.2.4.1 Window-Based Method

The window-based method is started by selecting the impulse response $h_N[n]$ as a symmetrically truncated version of the impulse response $h[n]$ of an ideal filter with frequency response $H(F)$. The impulse response of an ideal lowpass filter is $h[n]=2F_cSinc(2nF_c)$, where 'F_c' is the cutoff frequency. The symmetric truncation yields:

$$h_N[n] = 2F_cSinc(2nF_c), \quad |n| \le \frac{N-1}{2} \tag{1.1}$$

Truncation of the ideal impulse response $h[n]$ is equivalent to multiplication of $h[n]$ by a rectangular window $w[n]$ of length N (Figure 1.4). The spectrum of the windowed impulse response $h_W[n]=h[n]w[n]$ is the convolution of $H(F)$ and $W(F)$. Due to abrupt truncation of $h[n]$, the magnitude spectrum shows overshoot and ripples, referred to as the Gibbs phenomena. The tapered windows are used to reduce the Gibbs phenomena.

The spectrum of a window is approximated to an impulse. It should be confined to the narrowest mainlobe and the least energy in the sidelobes. There are many windows that have been developed with some optimality criterion. The trade-off is a compromise between the conflicting requirements of a narrow mainlobe (or a small transition width) and small sidelobe levels. Some windows are based on combinations of simpler windows. For example, the Hanning window is the sum of a rectangle and a cosine window; the Bartlett window is the convolution of two rectangle windows. Other windows are designed to emphasize certain desirable features. The Hanning window improves the high-frequency decay at the expense of a larger peak sidelobe level. The Hamming window minimizes the sidelobe level at the expense of a slower high-frequency decay. The Kaiser window has a variable parameter β that controls the peak sidelobe level.

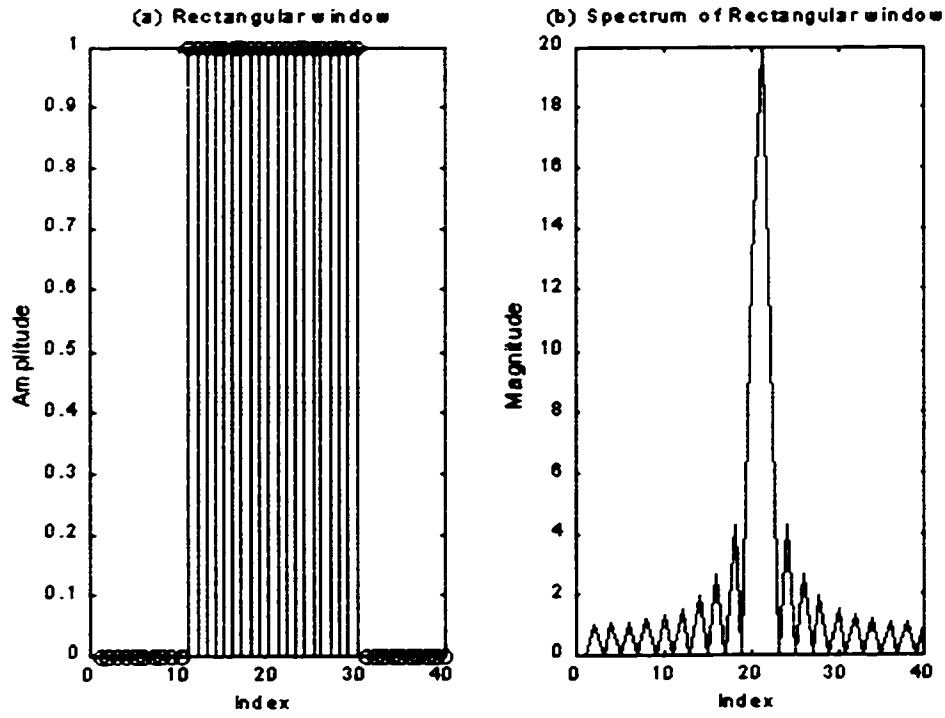


Figure 1.4: Rectangular window

The transition width of the windowed spectrum decreases with the filter length (N). No exact method establishes the minimum filter length that meets design specifications. However, empirical estimates are based on matching the given transition width specification F_T.

$$N = \frac{C}{F_T} \tag{1.2}$$

Where C ≈ constant per window.

For a given window, this relationship typically overestimates the smallest filter length. Therefore, the filter length could be decreased with satisfaction of the system design specification.

1.2.4.2 Frequency Sampling Method

The frequency sampling method is started with the required form for *H(F)*, and uses interpolation and DFT to obtain *h[n]*. It is more versatile since arbitrary frequency response forms can be easily solved.

To minimize the Gibbs effect in the design method, the sample values are varied gradually between jumps. This is equivalent to introducing a finite transition width. The selection of the sample values in the transition band can affect the response violently.

1.2.4.3 Optimal Constraint Iterative Method

To design an optimal linear-phase FIR filter requires minimizing the maximum error in the approximation. Optimal design of FIR filters is also based on a Chebyshev approximation. Such a design is expected to yield the smallest filter length and the same ripple in both the passband and the stopband.

There are three important concepts relevant to optimal design:

1) The error between the approximation *H(F)* and the desired response *D(F)* must be equal to the ripple. The error curve must show equal maximum and minimum values with alternating zero crossings. The greater the number of points where the error goes to zero (the zero crossings), the higher the order of the approximating polynomial and the longer the filter length.

2) The frequency response *H(F)* of a filter whose impulse response *h[n]* is a symmetric sequence can always be put in the form:

$$H(F) = Q(F)\sum_{n=0}^{M} \alpha_n \cos(2\pi nF) = Q(F)P(F) \tag{1.3}$$

Where *Q(F)* equals 1 (type 1), *cos(πF)* (type 2), *sin(2πF)* (type 3), or *sin(πF)* (type 4). *M* is related to the filter length *N* with *M=int[(N-1)/2]* (type 1,2,4) or *M=int[(N-3)/2]*. The α_n are related to the impulse response coefficients *h[n]*. The quantity *P(F)* may also be expressed as a power series in *cos(2πF)* (or as a sum of Chebyshev polynomials). If the selection of the α_n matches optimal constraints best, the *H(F)* is an optimal approximated to *D(F)*.

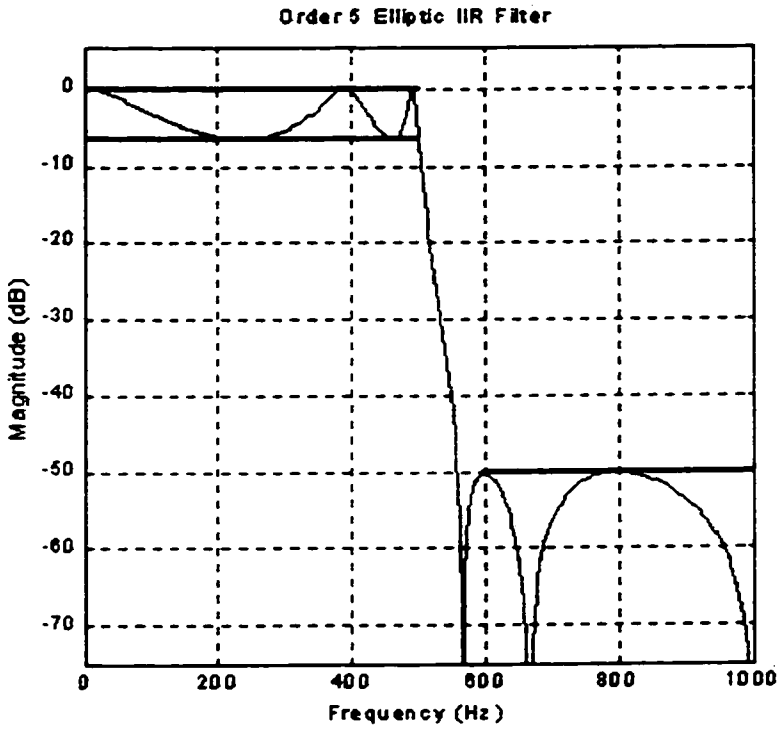


Figure 1.5: An elliptic IIR filter

3) The alternation theorem offers the hint to select the α_n. The alternation theorem points to minimize the maximum absolute error between *H(F)* and *D(F)*.

The disadvantages of this method are that the filter length must be estimated by empirical means and there is no control over the actual ripple.

1.2.5 General Elliptic Filter Design

Elliptic filters are based on the Jacobian elliptic-sine function. They yield the lowest filter order for given specifications by permitting ripples in both the

passband and stopband. For a given order, the filter exhibits the steepest transition region, although the ripple makes the most nonlinear phase and the longest delay characteristics. [5]

The elliptic approximation provides ripples in both bands by using rational functions in which numerator and denominator both display equal ripple properties. (Figure 1.5)

1.2.6 Half-Band Filters

If the normalized cutoff frequency, $F_c = 0.25$, then the FIR filter is a half-band lowpass/highpass FIR filter. (Figure 1.6. ***Note: All scaled frequencies in this thesis are based on sample frequency.***) The impulse response of an ideal lowpass filter is $h[n] = 2F_c Sinc(2nF_c) = 0.5 Sinc(0.5n)$. If n is an integer, then, a half-band FIR filter has an odd-length impulse response $h[n]$ whose alternate samples are zero. ($h[n] = 0$, when n is even.)

The peak values of the passband ripple and stopband ripple are equal in magnitude. The half-band filter design can be followed using the same steps as the window-based filter design.

Half-band filters will cause aliasing, since the lowpass and/or highpass filter are not ideal filters when the cutoff frequency $F_c = 0.25$.

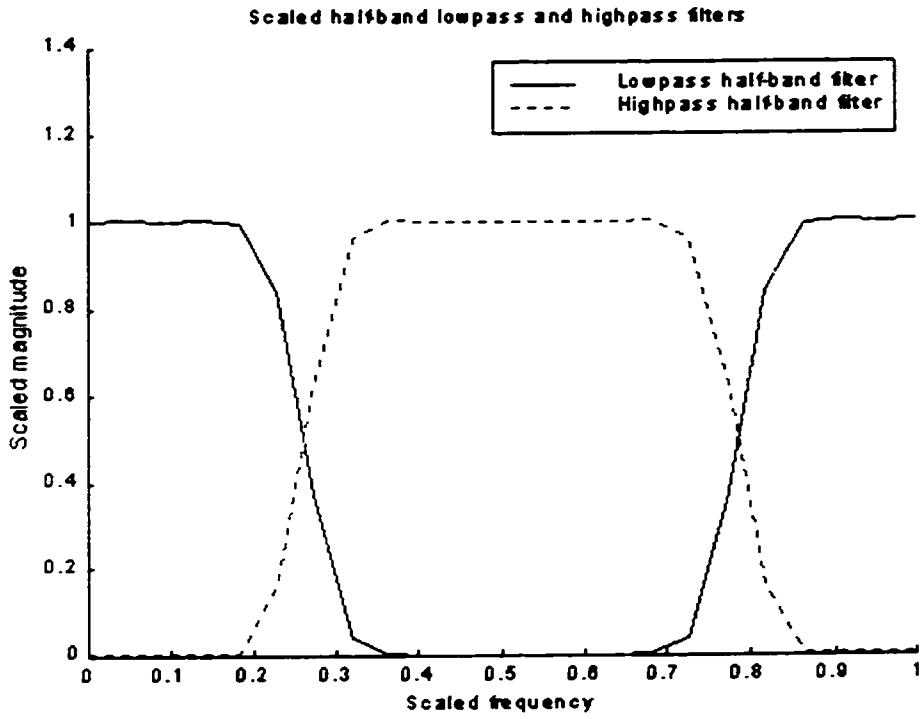


Figure 1.6: Scaled half-band lowpass and highpass filters

Chapter 2

THE GENERAL FIR FILTER USING JACOBIAN FUNCTION '*SN*'

2 Introduction

In this thesis, a special Jacobian elliptic-sine function is used. The shifted *Sn(u,m)*, with argument '*u*' in the frequency domain, is treated as a real magnitude filter with zero-phase shift linear phase.

2.1 Jacobian Elliptic Functions

A doubly periodic meromorphic function is called an elliptic function. It can also be defined with respect to certain integrals. Thus

$$u = \int_0^{\varphi} \frac{d\theta}{\left(1 - m\sin^2\theta\right)^{1/2}} \tag{2.1}$$

Where the angle φ is called the amplitude. It is defined as:

$$Sn(u) = \sin\varphi \tag{2.2}$$

Note: Since *m* is the parameter, *Sn(u)* can be redefined as *Sn(u,m)*. *K* and *K'* are called quarter-period here:

$$Sn(u,m) = \frac{2\pi}{\sqrt{m}K}\sum_{n=0}^{\infty}\frac{q^{(n+1)/2}}{1-q^{2n+1}}\sin\left[(2n+1)\frac{\pi u}{2K}\right]$$

$$K = \int_0^{\pi/2}\frac{d\theta}{\sqrt{1-k^2\sin^2\theta}}; K' = \int_0^{\pi/2}\frac{d\theta}{\sqrt{1-k'^2\sin^2\theta}} \tag{2.3}$$

$$q = e^{\frac{-\pi K'}{K}}; m = k^2; k^2 + k'^2 = 1$$

The Jacobian elliptic sine, *Sn(u,m)* is illustrated in Figure 2.1.

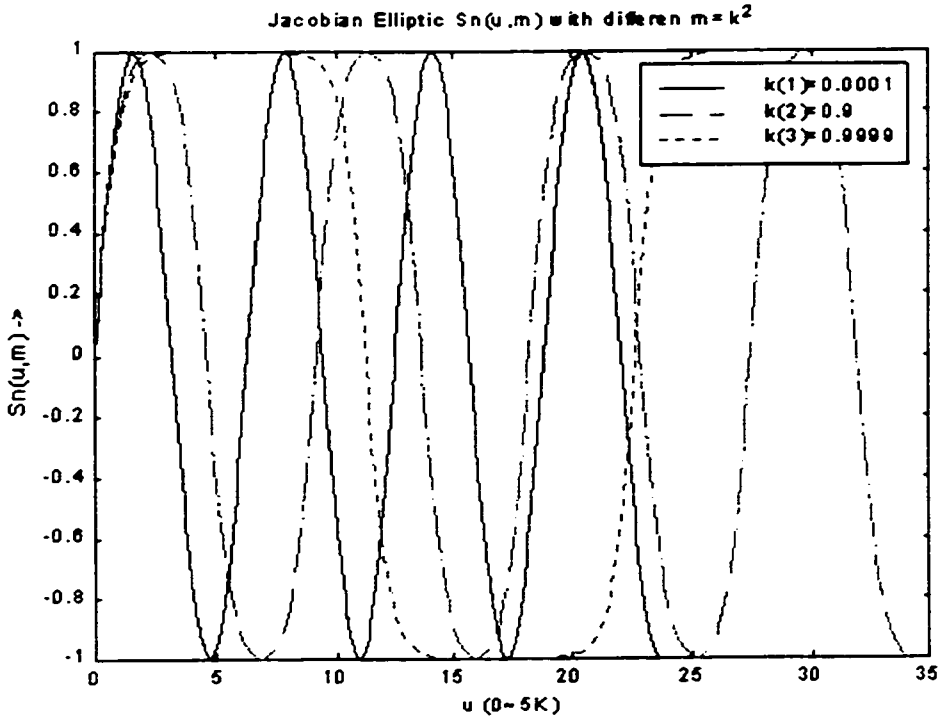


Figure 2.1: The Jacobian elliptic sine function

The Jacobian elliptic sine function behaves much like the trigonometric sine, except that it is flattened and elongated. The degrees of elongation (period) and flatness depend on the selection of parameter *m*. For *m=0*, *Sn(u,m)* is

identical to $sin(\phi)$. For a small u, $Sn(u,m)$ closely follows the sine function, and with increasing m, it becomes more flattened and elongated, reaching to an infinite period when $m=1$. This ability for $Sn(u,m)$ to change shape by changing m is what provides the means to characterize the function Sn.

2.2 A New Elliptic Filter Design Procedure

The new design procedure attempts to find the lowpass filter coefficients of cutoff frequency ω_c; then the filter coefficients are converted via IFFT.

According to the Fourier Transform theory, the number of frequency points is based on the number of input samples in the time domain. If the sampling frequency is N, the Nyquist frequency will be $N/2$. The cutoff frequency can be normalized based on the Nyquist frequency. It is between '0' to '1'. The cutoff frequency is a percentage of the Nyquist frequency. For example, if the cutoff frequency is 0.6 and the whole samples (N) are 100, the cutoff frequency point should at the 60^{th} point of the frequency domain. Figure 2.2 is the sketch map of this method. For displaying clearly, the scaled magnitude is shifted a little bit in this figure.

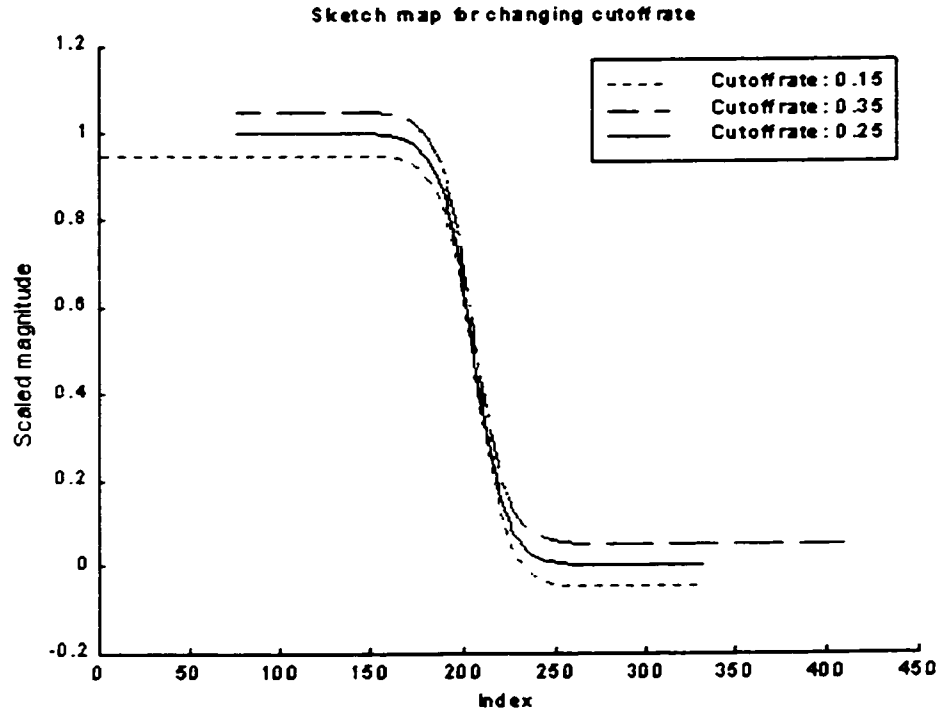


Figure 2.2: The method for changing cutoff frequency

Let us assume that those input data are generated based on the two quarter periods of corresponding weighted and shifted elliptic function *0.5[Sn(u+K,m)+1]* in frequency domain (Figure 2.3). In addition, their sequence number is fixed and the original cutoff frequency is *0.25*, a quarter of the sampling frequency or half of the Nyquist frequency. The cutoff frequency still can be adjusted: '*0*'s can be added after the sequence to make the cutoff frequency decrease; '*1*'s can be added in front of the sequence to make the cutoff frequency increase.

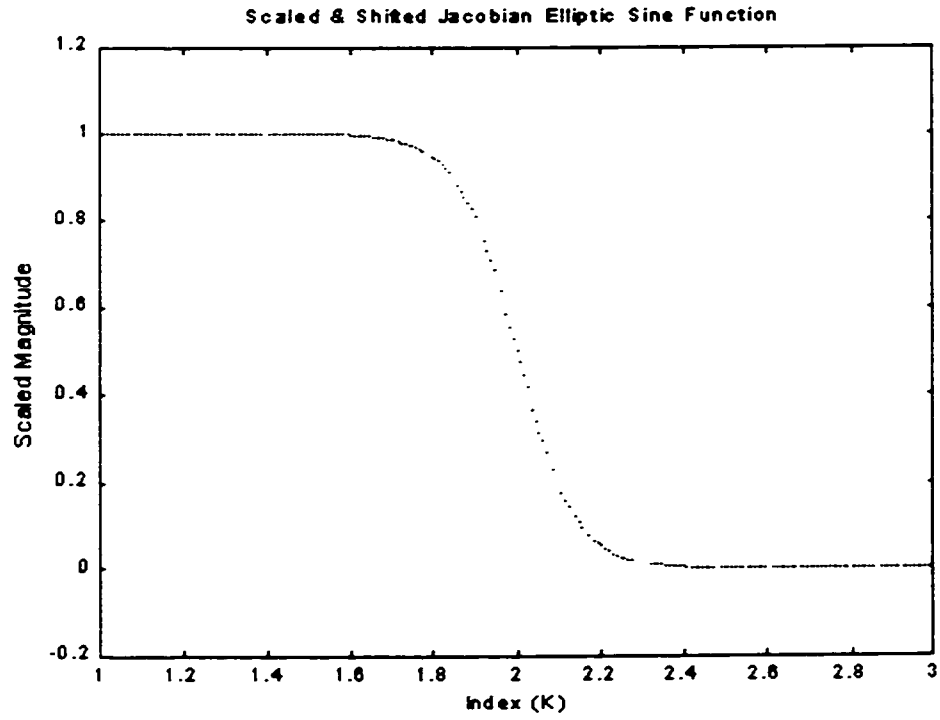


Figure 2.3: Scaled and shifted elliptic sine function ($m \rightarrow 1$)

2.2.1 Filter Design based on 'Sn'

Elliptic functions have many applications in digital signal processing, such as elliptic filters, noise modeling, approximations, etc. One of the attractive characteristics of these functions is the fast fall-off of their coefficients in their expansion series in terms of the Jacobian elliptic function *Sn(u,m)* function.

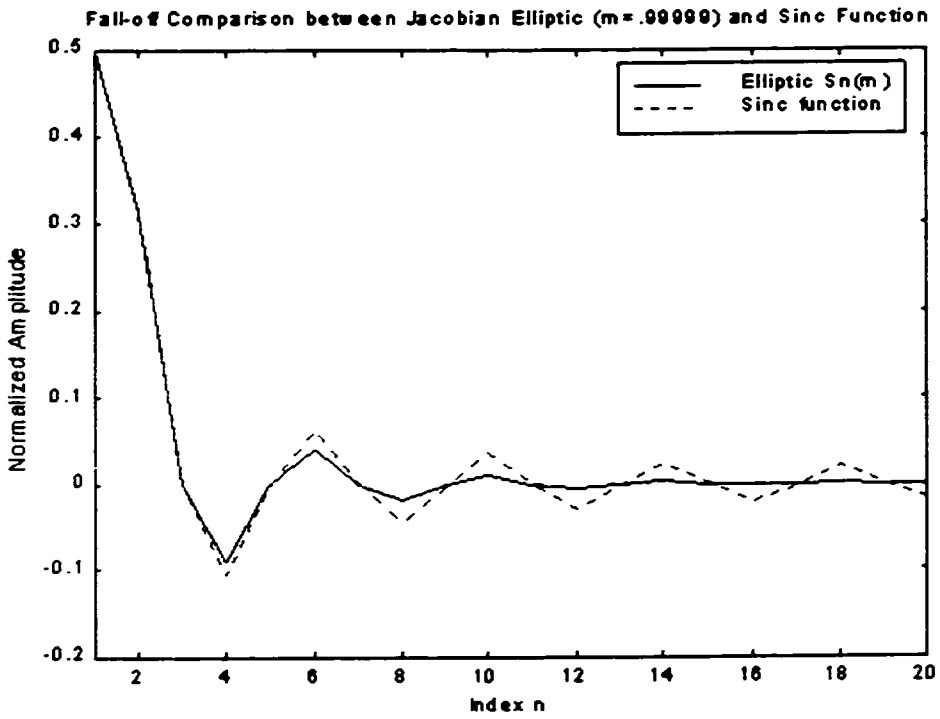


Figure 2.4: Coefficients of the elliptic Sn and Sinc function

For example, Figure 2.4 shows the coefficients of *Sn* function and *Sinc* function as n increases. It shows that the rate of decreasing *Sn(u,m)* coefficients for $m=0.99999$ is much more than in the Sinc function.

2.2.2 Design Procedure

The new design procedure is listed as follows:

1) Select the modulus factor m for the sharpness of the transition band.

2) Select the cutoff frequency ω_c of the filter and the filter order N.

3) Append zeros to confirm the input cut-off frequency when it is necessary.

4) Make an odd symmetry sequence of input data in the frequency domain.

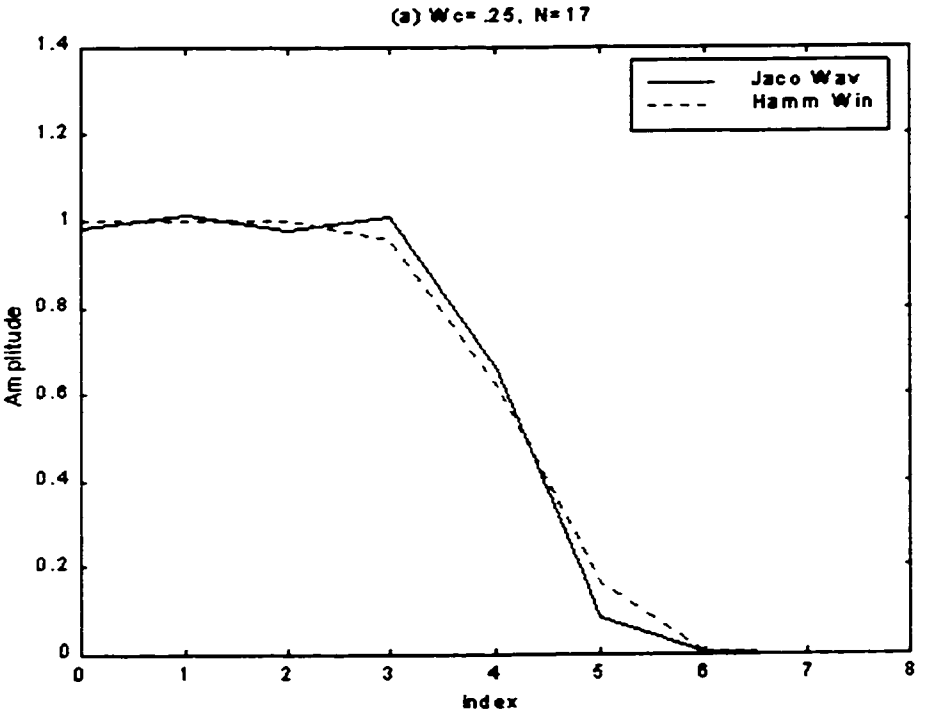
5) Use IFFT to obtain the impulse response filter samples in the time domain.

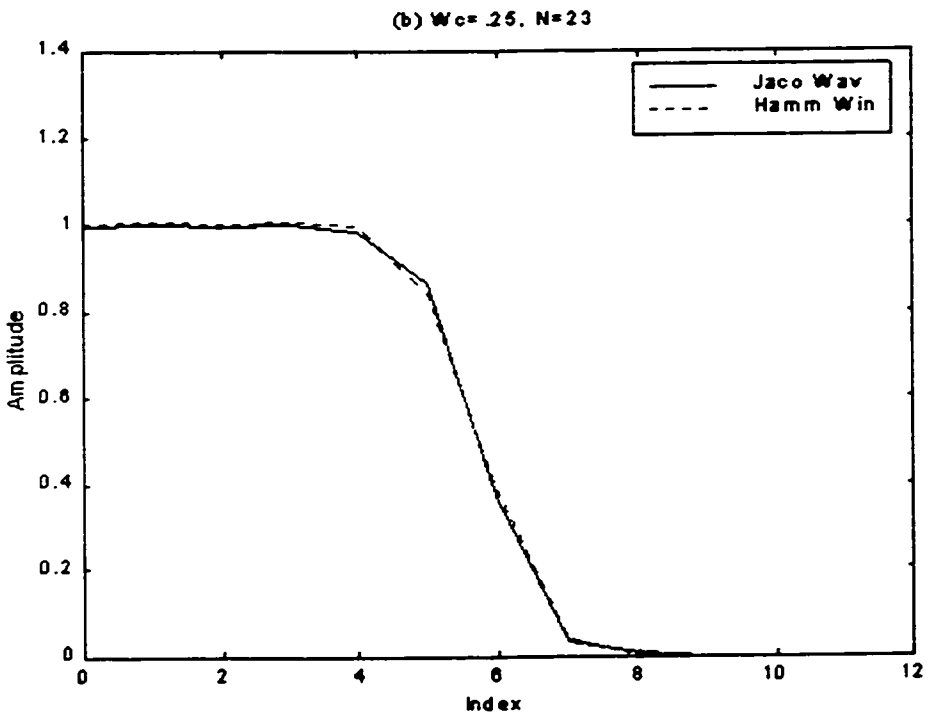
6) Truncate these samples in the time domain based on the filter order (N) requested. These samples can be treated as filter coefficients.

7) Check if the finite impulse response possesses the desired cutoff frequency and linear phase characteristic.

2.2.3 Compare the New Filter to the Normal Window-Based Filter

The simulation programs are written in MATLAB (version 5.1) on an IBM PC compatible computer. The results obtained from this program can be found in the four graphs of Figure 2.5, where 'W_c=*0.25*' means the cutoff frequency is a quarter of the sampling frequency and N is the filter order. These figures consist of various designs of FIR filters compared with the windowing design using a Hamming window. Different values of filter order show typical results. The solid line is the filter response using this new method (*m=0.99999* in *Sn(u,m)* function).The dashed line is the filter response using a Hamming window.


(a) Wc= .25, N=17
Jaco Wav
Hamm Win
Amplitude
Index
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
1
2
3
4
5
6
7
8


(b) Wc= .25, N=23
Jaco Wav
Hamm Win
Amplitude
Index
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
2
4
6
8
10
12

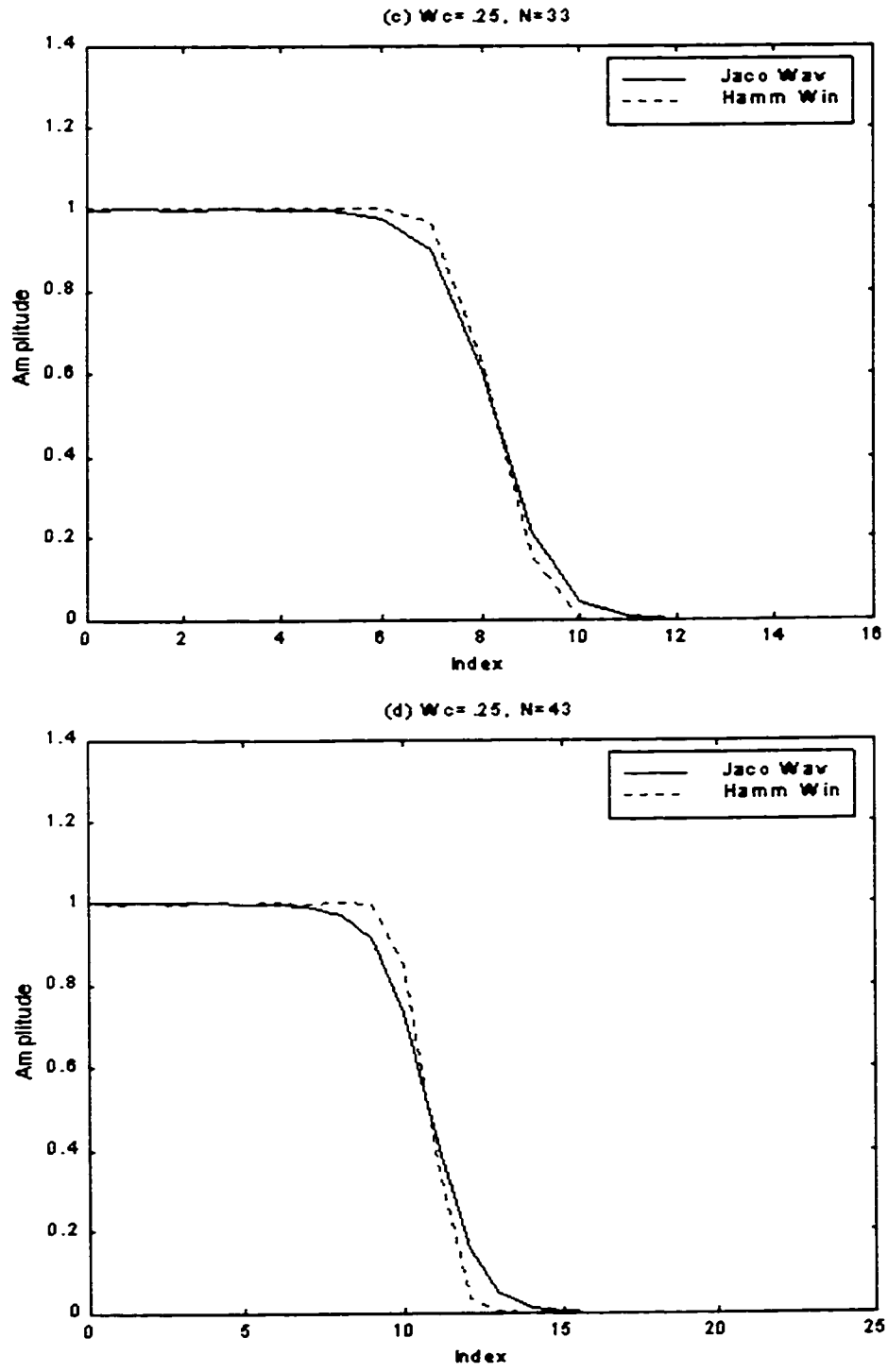


Figure 2.5: FIR filter design using new method versus Hamming window

2.3 Conclusions

The advantage of the proposed method is that the roll-off rate of impulse response coefficients for a modified *Sn* function is very fast. Thus, the total number of filter taps is one of the most important facts. However, for a sharp transition band, the number of filter coefficients should be increased. The disadvantage of the proposed method is that there is not an analytic method for specifying the pass-band ripple as a function of the truncation length of the impulse response coefficients.

If this new filter design method is used in half-band filter systems, another advantage is shown. Since the even coefficients of half-band filter are '*0*', its realization requires only about half the number of multipliers. In addition, most of the new filter coefficients are 2^n (n is negative integer), so most of these 'multiply' operations are only needed to do the 'shift' operations.

Chapter 3

QMF WAVELET BASED ON *SN* FILTER BANKS

3 Introduction

Quadrature Mirror Filter Banks (QMFB) are often used in wavelet analysis. They may be used for Perfect Reconstruction (PR). A new kind of filter, the Quasi-Dyadic filter, is provided and used in the QMFB.

3.1 Background

Signal analysis plays an important role in modern technologies. Human speech music and many other types of signals have to be efficiently encoded, compressed, reconstructed, etc.

The best known of the signal analysis tools is Fourier analysis. Fourier theory shows that a signal can be broken into a series (possibly infinite) of sinusoids of different frequencies. The sum of this series is referred to as a Fourier expansion. Fourier Transform (FT) is a mathematical technique for transforming the signal from the time domain to the frequency domain. The big disadvantage of a Fourier expansion is that it has only frequency resolution

and no time resolution. This means it is impossible to determine when the frequencies are present.

To correct this deficiency, Short-Time Fourier Transform (STFT) was developed by Dennis Gabor (1946). The idea is to cut the signal of interest into several small sections (windowing signals) and then analyze the sections separately. It maps signals into a two-dimensional function of time and frequency. The drawback is that once the particular size of the time window is selected, that window is the same for all frequencies. Cutting the signal corresponds to a convolution between the signal and the cutting window. The problem here is how to choose the window size. Suppose a very small window is chosen for analysis. Since convolution in the time domain is identical to multiplication in the frequency domain and the Fourier transform of a pulse contains all possible frequencies, the frequency components of the signal will be smeared all over the frequency axis. In fact, this situation is in time resolution but not frequency resolution. That is Heisenberg's uncertainty principle, which, in signal processing terms, states that it is impossible to know the exact frequency and the exact time of occurrence of this frequency in a signal. ***In other words, a signal can simply not be represented as a point in the time-frequency space.*** [15]

The next logical step is to find a windowing technique with variable-sized regions. The *wavelet transform* or *wavelet analysis* is probably the most recent solution for this objective. It overcomes the shortcomings of FT and STFT. In wavelet analysis, the use of a fully scalable modulated window solves the signal-cutting problem. The window is shifted along the signal and the spectrum is calculated for every position. This process is repeated many times with a slightly shorter (or longer) window for every new cycle. The result will be a collection of time-frequency representations of the signal including all different resolutions. That is called multiresolution analysis.

In the case of wavelets we normally do not speak about time-frequency representations but about time-scale representations, scale being in a way the opposite of frequency, because the term frequency is reserved for the Fourier transform.

3.1.1 History of Wavelets [9][12][13][14]

1) The term, 'wavelet', was provided by Alfred Haar in 1909.

2) Wavelet applications were introduced by J. Morlet, a French geophysicist for the analysis of seismic data in 1980.

3) Ingrid Danbechies provided a major breakthrough by constructing families of orthonormal wavelets with compact support in 1988.

4) An efficient way to implement multi-resolution signal decomposition using filters was developed by Mallat in 1988.

5) Mallat related QMFB to the mathematical theory of wavelets.

6) The Mallat algorithm is in fact a classical scheme known in the signal processing community as two-channel filter bank coding.

7) The related algorithms are used in decomposition and reconstruction of signals. Wavelet was introduced by Morlet in 1980 when he analyzed the data of earthquakes. Mallat produced a new kind of fast wavelet decomposition and reconstruction algorithm. The algorithm for discrete wavelet transfer (DWT) is known as Quadrature Mirror Filter (QMF).

3.1.2 Wavelets Analysis and Applications

A wavelet is a waveform of effectively limited duration that has an average value of zero. So there are many wavelets, but most of them tend to be irregular and asymmetric.

The wavelets are generated from a basic wavelet $\psi(t)$, which is called the mother wavelet, by scaling and translation:

$$\psi_{s,\tau}(t)=\frac{1}{\sqrt{s}}\psi\left(\frac{t-\tau}{s}\right) \tag{3.1}$$

Where s is the scale factor, τ is the translation factor and the factor $s^{-1/2}$ is for energy normalization across the different scales.

Wavelet analysis is the breaking up of a signal into shifted and scaled versions of the original (or *mother*) wavelet. It is much like a Fourier Transform in that it can be used to substitute a signal for sine waves of various frequencies. Continuous Wavelet Transform (CWT) and Discrete Wavelet Transform (DWT) are mainly used in wavelet analysis.

The continuous wavelet transform or CWT can be formally written as:

$$\gamma(s,\tau) = \int f(t)\psi^*_{s,\tau}(t)dt \tag{3.2}$$

Where '*' denotes complex conjugation. This equation shows how a function $f(t)$ is decomposed into a set of basis functions $\psi_{s,\tau}(t)$, called the wavelets. The variables s and τ are the new dimensions, scale and translation, after the wavelet transform. Although we are not going to use it, for completeness sake, the inverse wavelet transform is given as follows:

$$f(t) = \iint \gamma(s,\tau)\psi(t)d\tau ds \tag{3.3}$$

The continuous wavelet transfer of a real signal is the result of signal and analysis wavelet convolution.

There is a correspondence between wavelet scales and frequencies:

1) If low scale *s* is chosen, the wavelet is compressed. The wavelet analysis can obtain the rapidly changing details of the incoming signals, which means the information hides in the high frequency sections of the incoming signal.

2) If high scale *s* is chosen, the wavelet is stretched. The wavelet analysis can obtain the slowly changing, approximate features of the incoming signals, which means the information hides in the low frequency sections of the incoming signal.

CWT can operate at every scale, and it is continuous in terms of shifting. Its analyzing wavelet is shifted smoothly over the full domain of the analyzed function during computation.

Table 3.1: Two-channel Filter Bank

Filters	Lowpass	Highpass
Decomposition (Analysis)	H_L	H_H
Reconstruction (Synthesis)	G_L	G_H

DWT is defined as choosing the wavelet scales and positions based on powers of two — so called *dyadic* scales and positions. Therefore, the analysis will be much more efficient and still accurate.

3.2 Analysis & Synthesis Filter Banks

If two or more than two filters are combined, it is called a filter bank. The idea is to separate the input signal in the frequency domain. A two-channel filter bank is usually used to explain the idea. Suppose a lowpass filter and a highpass filter split the incoming signal into two parts in the frequency domain. Those parts will be compressed and coded individually. They can be transmitted and the signal can be recovered. If the synthesis filter bank exactly recovers the input signal, it is called a Perfect Reconstruction (PR) filter bank.

Figures 3.1, 3.2 and 3.3 on the following pages show the analysis and synthesis sections of a two-channel filter bank. The four FIR filters are defined in Table 3.1.

3.2.1 Downsampling & Upsampling

A two-channel filter bank is presented in Figure 3.1, assuming the length of filters $H_L(z)$ and $H_H(z)$ are N. The length of each section $x[n]$ is N. The length of $v_L[n]$ and $v_H[n]$ becomes $2N$. Only N samples are expected in the following processing. The procedure converting $2N$ to N is downsampling (Figure 3.1).

It removes every other component in the data series. It is used in the analysis section of filter bank systems.

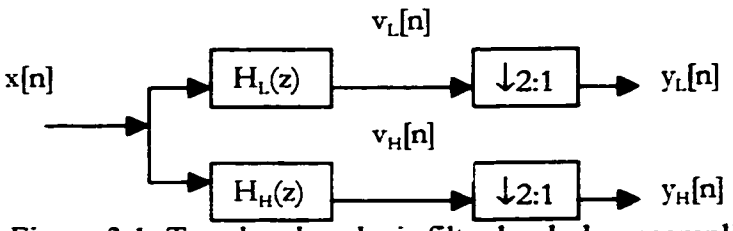


Figure 3.1: Two-band analysis filter bank downsampling

The synthesis section performs the inverse procedure of the analysis transformation for recovering the incoming signal. Here, in the following figure (Figure 3.2), the two sequences $y_L[n]$ and $y_H[n]$ are upsampled. After being filtered by filters $G_L(z)$ and $G_H(z)$, they are combined to provide an approximation $x^*[n]$ with respect to the original sequence, $x[n]$.

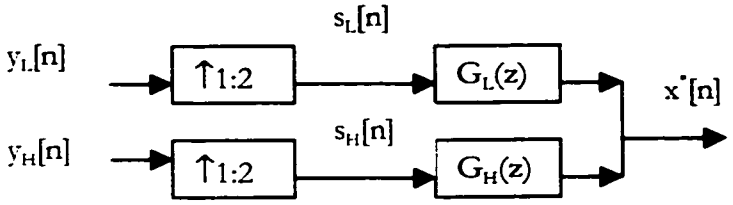


Figure 3.2: Two-band synthesis filter bank upsampling

For a purely exponential signal $x[k]=e^{jk\omega}$, the effect of downsampling is particularly clear. The kth component is $y[k]=e^{j2k\omega}$. This is a purely

exponential with a frequency 2ω. Frequencies are doubled by downsampling. But that does not mean $Y(2\omega)=X(\omega)$ or $Y(\omega)=X(\omega/2)$.

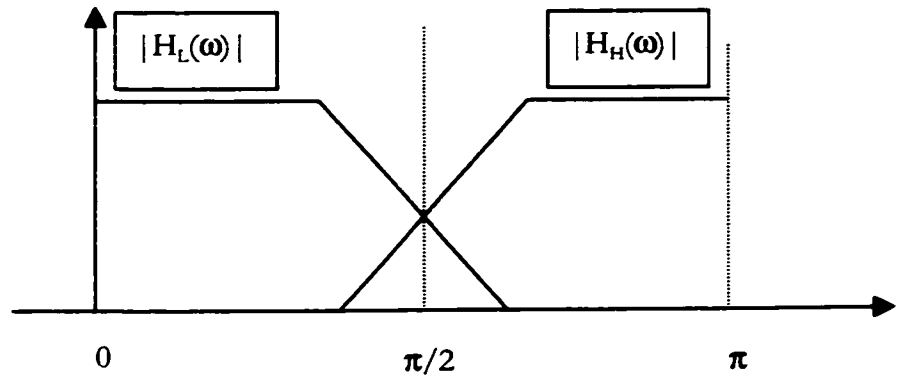


Figure 3.3: Two-channel filter bank frequency responses

Assume x' is another purely exponential with frequency $\omega+\pi$. Then $x'[k]=e^{jk(\omega+\pi)}$ is downsampled to $y'[k]=e^{j2k(\omega+\pi)}$. The factor $e^{j2k\pi}$ is always 1. This output is the same $e^{j2k\omega}$ as before. Adding π to the frequency only reverses the signs of the odd-numbered components of x'. Those components disappear in downsampling, and $y'[k]=y[k]$. The frequency 2ω appears by doubling ω, and it appears by doubling $\omega+\pi$. In other words, ω enters the downsampled vector v not only by doubling $\omega/2$ but also by doubling $\omega/2+\pi$. These two sources can be written as follows:

$$Y(\omega)=\frac{1}{2}\left[X\left(\frac{\omega}{2}\right)+X\left(\frac{\omega}{2}+\pi\right)\right] \tag{3.4}$$

And the result $X^{*}(\omega)$ can be presented as:

$$X^*(\omega) = Y(2\omega) = \frac{1}{2}[X(\omega) + X(\omega + \pi)] \tag{3.5}$$

Here $X(\omega+\pi)$ is the alias term.

3.2.2 Perfect Reconstruction

The goal of the analysis and synthesis is to find the perferct reconstruction solution. This means that the filter bank should be satisfied to:

$$G_L H_L + G_H H_H = I \tag{3.6}$$

It presents the reconstruction without delay. A perfect reconstruction without downsampling and upsampling can be written as

$$G_L(z)H_L(z) + G_H(z)H_H(z) = z^{-l} \tag{3.7}$$

where z^{-l} is an overall l-step delay. Because the sampling operators produce a change, the "no" distortion formula should be

$$H_L(z)G_L(z) + H_H(z)G_H(z) = 2z^{-l}. \tag{3.8}$$

The estimation output signal can be written as [14] :

$$X^*(z) = \frac{1}{2}\{[H_L(z)G_L(z) + H_H(z)G_H(z)]X(z) + [H_L(-z)G_L(z) + H_H(-z)G_H(z)]X(-z)\} \tag{3.9}$$

Since the perfect reconstruction should have no alias and no distortion, the condition of the two-channel filter bank PR is:

$$\begin{bmatrix} G_L(z) & G_H(z) \end{bmatrix} \begin{bmatrix} H_L(z) & H_L(-z) \\ H_H(z) & H_H(-z) \end{bmatrix} = \begin{bmatrix} 2z^{-l} & 0 \end{bmatrix} \tag{3.10}$$

This matrix is sometimes called a modulation matrix. It involves the responses $H_k(z)$ and their alias terms $H_k(-z)$. It can be expanded to an $M \times M$ matrix for an M-channel bank.

3.2.3 Perfect Reconstruction Filter Banks Design

The alias must be canceled for a PR system. This can be presented as follows:

$$G_L(z) = H_H(-z) \quad G_H(z) = -H_L(-z) \tag{3.11}$$

This selection automatically satisfies the modulation matrix.

And if the equations are defined:

$$P_L(z) = G_L(z)H_L(z) \quad P_H(z) = G_H(z)H_H(z) \tag{3.12}$$

then another part of the modulation matrix can be rewritten as:

$$G_L(z)H_L(z) + G_H(z)H_H(z) = G_L(z)H_L(z) - G_L(-z)H_L(-z) = P_L(z) - P_L(-z) = 2z^{-l} \tag{3.13}$$

Then the design of a two-channel PR filter bank may be reduced to three steps:

1) Design a lowpass filter P_L

2) Factor P_L into $G_L H_L$

3) Find G_H and H_H

3.2.4 Quadrature Mirror Filters (QMF)

The term QMF refers to a particular selection of filters that are related by spatial shifting and frequency modulation.

If the filters are defined:

$$\begin{aligned} H_L(\omega) = H(\omega) &\quad \Leftrightarrow \quad h(n) \\ H_H(\omega) = H(\omega+\pi) &\quad \Leftrightarrow \quad (-1)^n h(n) \end{aligned} \tag{3.14}$$

Its 'z' domain definition is:

$$H_H(z) = H_L(-z) \tag{3.15}$$

This system is called a QMF filter bank (Figure 3.4). Their features are:

- Only the low pass filter h(n) needs to be designed
- The transfer from $X(\omega)$ to $X^*(\omega)$ is

$$X^*(\omega) = \frac{1}{2}\left[H(\omega)^2 - H(\omega+\pi)^2\right]X(\omega) \tag{3.16}$$

If the symmetric finite impulse response (FIR) filter is selected:

$$H(\omega) = H_g(\omega)e^{-j\omega\frac{N-1}{2}} \tag{3.17}$$

then the transfer function is:

$$\begin{aligned}\frac{X^*(\omega)}{X(\omega)} &= H(\omega)^2 - H(\omega+\pi)^2 \\ &= H_g^2(\omega)e^{-j\omega(N-1)} - H_g^2(\omega+\pi)e^{-j\omega(N-1)}e^{-j\pi(N-1)} \\ &= \left[H_g^2(\omega) - (-1)^{N-1}H_g^2(\omega+\pi)\right]e^{-j\omega(N-1)}\end{aligned} \tag{3.18}$$

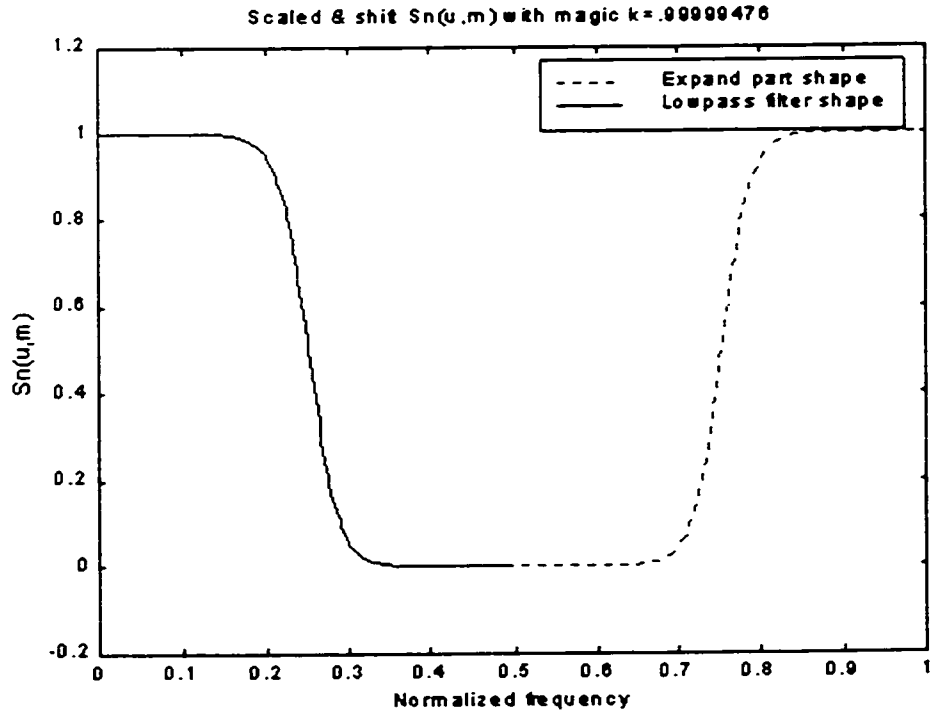


Figure 3.4: Lowpass filter and its QMF

For even values of $N-1$, this leads to a phase jump at $\omega=\pi/2$:

$$\frac{X^*(\omega)}{X(\omega)} = \begin{cases} H_g^2(\omega)e^{-j\omega(N-1)} & \omega \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \\ -H_g^2(\omega+\pi)e^{-j\omega(N-1)} & |\omega| \in \left[\frac{\pi}{2}, \pi\right] \end{cases} \tag{3.19}$$

So the QMF filter taps (coefficients) is usually chosen as even. For the PR requirement, the ideal transfer function is equal to *'1'*.

$$H_g^2(\omega) + H_g^2(\omega+\pi) = 1 \tag{3.20}$$

The above formula cannot be realized if $N>2$. Therefore, most of QMF low pass filters have been designed that result in small deviations for the unit transfer.

3.2.5 Polyphase Decomposition

Polyphase Decomposition was provided by Dr. M. Bellanger in 1970. [9]

Assume a transform of a digital filter is:

$$\begin{aligned} H(z) = &\left[\cdots + h(-4)z^4 + h(-2)z^2 + h(0) + h(2)z^{-2} + h(4)z^{-4} + \cdots\right] \\ &+ z^{-1}\left[\cdots + h(-3)z^3 + h(-1)z + h(1)z^{-1} + h(3)z^{-3} + \cdots\right] \end{aligned} \tag{3.21}$$

If the following equations are defined:

$$E_1(z) = \sum_{n=-\infty}^{\infty} h(2n)z^{-n} \quad E_2(z) = \sum_{n=-\infty}^{\infty} h(2n+1)z^{-n} \tag{3.22}$$

Then (3.22) can be rewritten as:

$$H(z) = E_1(z^2) + z^{-1}E_2(z^2) = \sum_{k=0}^{2-1} z^{-k}E_k(z^2) \tag{3.23}$$

where $E_1(z^2) = \sum_{k=-\infty}^{\infty} h(k)z^{-k}$ and $E_2(z^2) = \sum_{k=-\infty}^{\infty} h(2k+1)z^{-k}$. (3.24)

For M channels, the general equation is: $H(z) = \sum_{k=0}^{M-1} z^{-k}E_k(z^M)$ (3.25)

3.3 Quasi-Dyadic FIR Filter Bank

If a special m is selected in *Sn(u,m)*, most of its non-zero coefficients may become 2^n (n is negative integer). The QMF wavelet filter bank which is using this *Sn(u,m)* function with special 'm' is a Quasi-Dyadic FIR filter bank.

3.3.1 Special Character in Jacobian Elliptic Sn(u,m)

The Jacobian elliptic sine function *Sn(u,m)* can be rewritten as:

$$\begin{aligned} Sn(u,m) &= \frac{2\pi}{\sqrt{m}K}\sum_{n=0}^{\infty}\frac{q^{(n+1)/2}}{1-q^{2n+1}}\sin\left[(2n+1)\frac{\pi u}{2K}\right] = S\sum_{n=0}^{\infty}A\sin[B] \\ K &= \int_0^{\pi/2}\frac{d\theta}{\sqrt{1-k^2\sin^2\theta}};K' = \int_0^{\pi/2}\frac{d\theta}{\sqrt{1-k'^2\sin^2\theta}} \\ q &= e^{\frac{-\pi K'}{K}};m = k^2;k^2+k'^2 = 1 \end{aligned} \tag{3.26}$$

where 'S' is the scale of *Sn(u,m)*, 'B' stands for the harmonics and 'A' is the scale of each harmonic. It is found that $A\times 2^n$ (n is the index) almost approximates to a constant (except the first several values) when 'k' equals ***0.9999947610549319*** or 'm' equals ***0.99998952213731035*** [3]. Table 3.2

provides $A \times 2^n$ values. Figure 3.5 shows the *Sn(u,m)* with the magic value of '*k*' comparing to other values of '*k*'.

Table 3.2: $A \times 2^n$ values based on *k=0.9999947610549319*

Index (n)	Value	Value	Value	Value	Value	Value	Value	Value
1~8	1.4142	0.8081	0.7299	0.7127	0.7085	0.7075	0.7072	0.7071
9~16	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071
17~24	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071
25~32	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071
33~40	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071
41~48	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071
49~56	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071
57~64	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071	0.7071

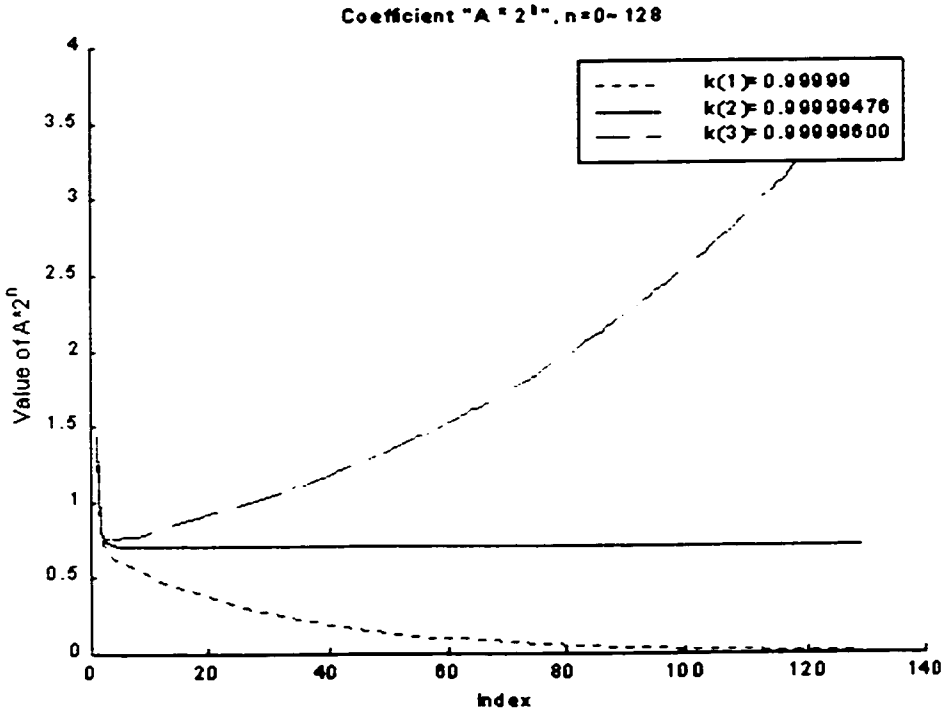


Figure 3.5: Magic value 'k' in *Sn(u,m)* function

It can be assumed that the magic character is kept and hidden in the IFFT result of *Sn(u,m)*. A half-band lowpass Quasi-Dyadic FIR filter (Figure 3.6) is designed to prove it.

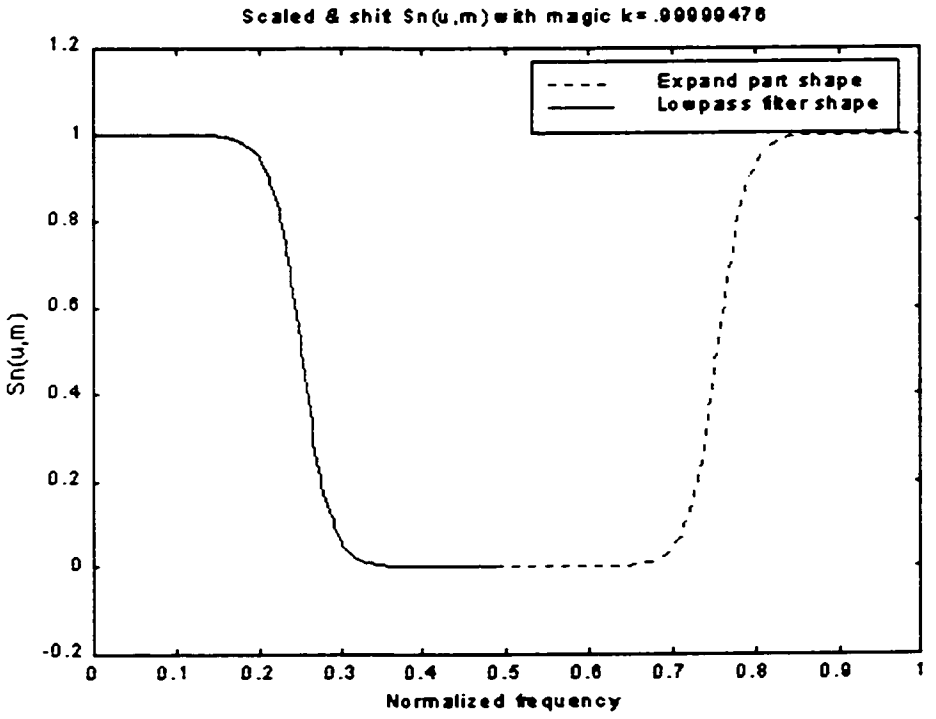


Figure 3.6: A half-band lowpass Quasi-Dyadic FIR filter

Quasi-Dyadic FIR coefficients can be obtained by using IFFT to *Sn(u,m)* function. To prove the above idea, 2^n (*n* is negative integer) is multiplied by the coefficients. Figure 3.7 presents the result. Obviously, the coefficients have the following useful features that are expected:

1) The coefficients are zero alternation except the first one.

2) The even coefficients are zero except the first one.

3) Most of the odd coefficients are 2^n (n is negative integer).

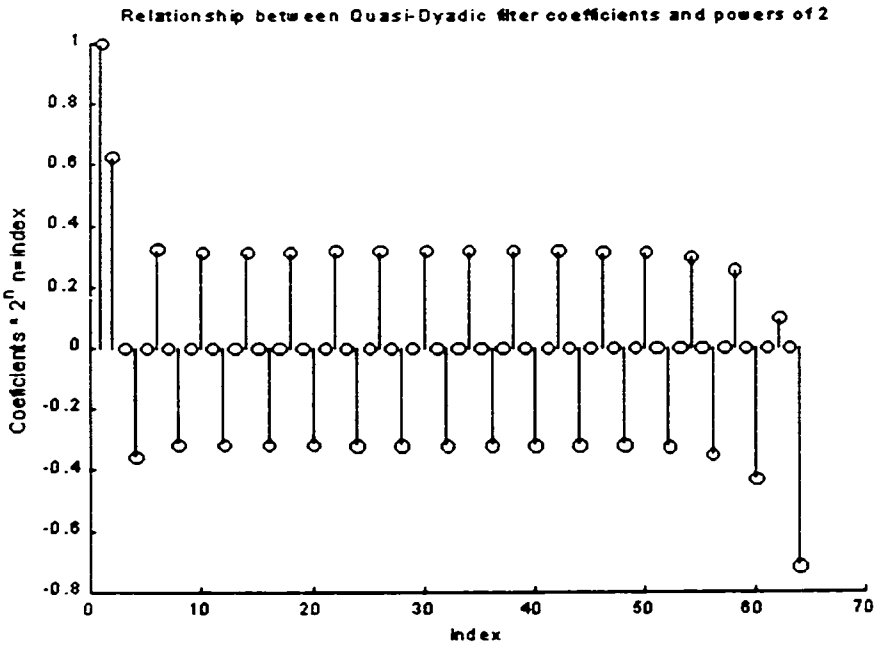


Figure 3.7: Quasi-Dyadic filter coefficients multiply by 2^n, 'n = index/2'

Table 3.3 shows the Quasi-Dyadic filter coefficients, coefficients multiplied by 2^n (n=*index/2*, because each even coefficient is '*0*' except the first one) and their normalized values based on *0.31716*. The third feature disappears after index n=*47*. Since the coefficients are symmetric, the Quasi-Dyadic filter order should not be more than *94*. In addition, the normalized values approximating to *'–1 0 1'* between n=*4* to n=*47* (4th and 8th columns) are useful in optimizing the filter calculation.

Table 3.3: The Quasi-Dyadic filter coefficients, coefficients multiply by 2^n (n=index/2) and their normalized value

Index (n)	coef	coef*2^n	coef*2^n/0.31716	index (n)	coef	coef*2^n	coef*2^n/0.31716
1	5.00E-01	1.0000	3.15298272	2	3.12E-01	0.6241	1.96789619
3	-1.12E-16	0.0000	0.00000000	4	-8.93E-02	-0.3570	-1.12564235
5	2.64E-17	0.0000	0.00000000	6	4.04E-02	0.3230	1.01841343
7	-1.31E-16	0.0000	0.00000000	8	-1.97E-02	-0.3160	-0.99630813
9	-2.38E-17	0.0000	0.00000000	10	9.84E-03	0.3148	0.99243835
11	-6.82E-17	0.0000	0.00000000	12	-4.92E-03	-0.3149	-0.99288471
13	2.58E-17	0.0000	0.00000000	14	2.46E-03	0.3154	0.99429570
15	1.53E-17	0.0000	0.00000000	16	-1.23E-03	-0.3158	-0.99584466
17	-5.38E-17	0.0000	0.00000000	18	6.18E-04	0.3163	0.99731070
19	-1.42E-16	0.0000	0.00000000	20	-3.09E-04	-0.3167	-0.99865362
21	-9.84E-17	0.0000	0.00000000	22	1.55E-04	0.3171	0.99983831
23	-1.49E-16	0.0000	0.00000000	24	-7.75E-05	-0.3174	-1.00089561
25	-3.77E-17	0.0000	0.00000000	26	3.88E-05	0.3177	1.00176388
27	-1.53E-16	0.0000	0.00000000	28	-1.94E-05	-0.3180	-1.00254472
29	-1.72E-16	0.0000	0.00000000	30	9.71E-06	0.3181	1.00305817
31	-1.59E-16	0.0000	0.00000000	32	-4.86E-06	-0.3183	-1.00361748
33	-1.27E-16	0.0000	0.00000000	34	2.43E-06	0.3183	1.00386742
35	-1.57E-16	0.0000	0.00000000	36	-1.21E-06	-0.3185	-1.00419071
37	-1.41E-16	0.0000	0.00000000	38	6.07E-07	0.3182	1.00343109
39	-1.03E-16	0.0000	0.00000000	40	-3.04E-07	-0.3186	-1.00453684
41	-1.22E-16	0.0000	0.00000000	42	1.52E-07	0.3177	1.00183095
43	-1.62E-16	0.0000	0.00000000	44	-7.60E-08	-0.3189	-1.00558787
45	-1.04E-16	0.0000	0.00000000	46	3.77E-08	0.3163	0.99713298
47	-1.37E-16	0.0000	-0.00000001	48	-1.91E-08	-0.3205	-1.01052463
49	-2.39E-16	0.0000	-0.00000003	50	9.30E-09	0.3119	0.98343739
51	-1.91E-16	0.0000	-0.00000004	52	-4.87E-09	-0.3268	-1.03026218
53	-1.90E-16	0.0000	-0.00000008	54	2.22E-09	0.2983	0.94046263
55	-2.52E-16	0.0000	-0.00000021	56	-1.30E-09	-0.3497	-1.10250205
57	-2.20E-16	0.0000	-0.00000037	58	4.71E-10	0.2531	0.79797764
59	-2.02E-16	0.0000	-0.00000068	60	-4.01E-10	-0.4308	-1.35816090
61	-3.37E-16	0.0000	-0.00000228	62	4.61E-11	0.0989	0.31185048
63	-2.51E-16	0.0000	-0.00000340	64	-1.66E-10	-0.7141	-2.25144103

Figure 3.8 illustrates the symmetric coefficients of the Quasi-Dyadic FIR filter with 23 taps.

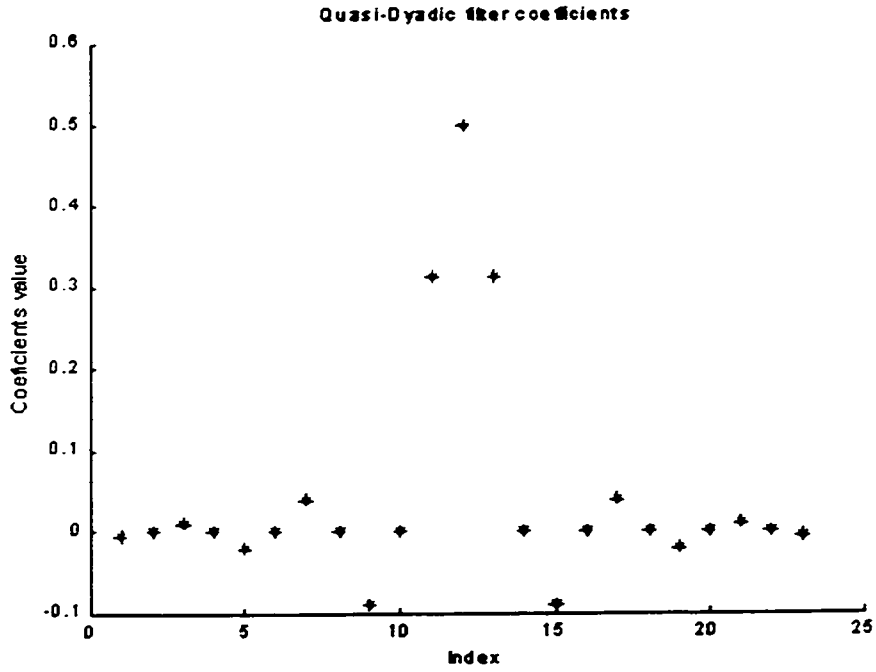


Figure 3.8: Quasi-Dyadic FIR coefficients

3.3.2 *Quasi-Dyadic QMFB Analysis*

Assume the alias is already canceled in the Qusasi-Dyadic QMFB by specially selecting the filter coefficients based on equation (3.11). The analysis filter bank is selected as QMFB, which is represented by equations (3.14) or (3.15). The necessary conditions for a PR system are equations (3.8) and (3.11). But according to the design method of the Quasi-Dyadic FIR filter, only (3.11) can be satisfied while (3.8) may or may not be satisfied.

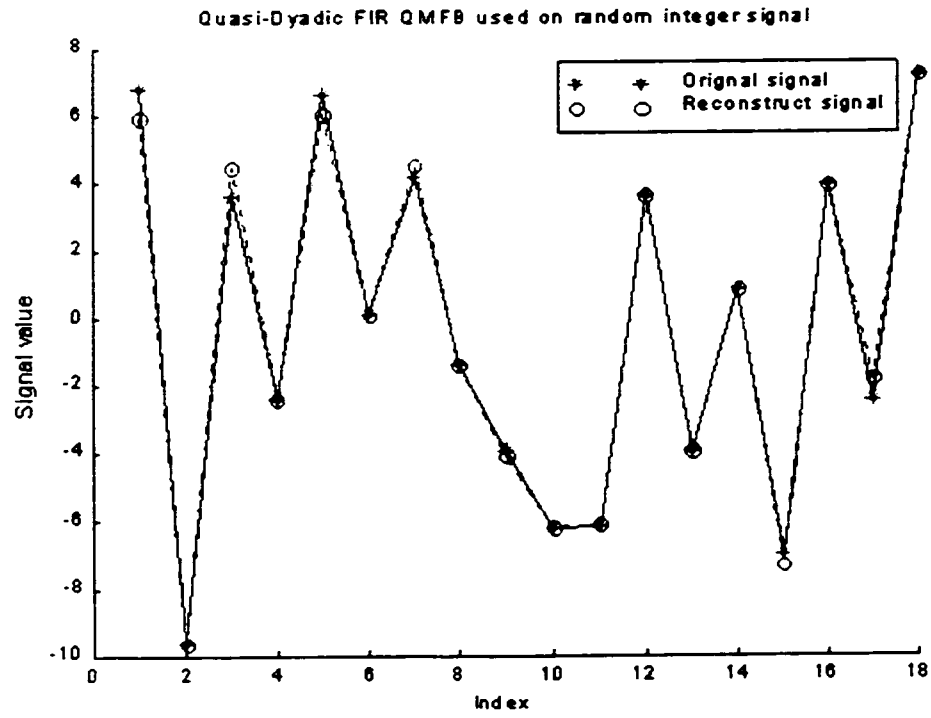


Figure 3.9: Quasi-Dyadic Fir QMFB based on random signal

The simulation results show that the Quasi-Dyadic filter can do the perfect reconstruction on the odd input samples while there are distortions on the even

input samples. (Figure 3.9. ***Note: The index used in Matlab calculation starts from '1' not '0'. This changes the odd and even order in all figures in this thesis.***)

This interesting result can be deduced according to polyphase decomposition, Noble Identity, and the special coefficients of the Quasi-Dyadic filter. The following paragraphs present this deduction. Figure 3.10 illustrates the Noble Identity, where '*M*' and '*L*' are downsampling and upsampling orders

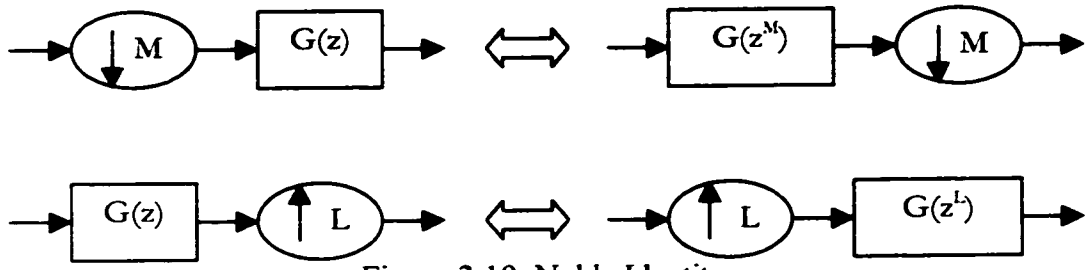


Figure 3.10: Noble Identity

It illustrates:

1. Any decimation process behind a filter can be moved in front of the filter with suitable downsampled coefficients

2. Any zero-inserted process in front of a filter can be moved behind the filter with suitable downsampled coefficients.

Assume there is a simple PR system (Figure 3.11). $H_L(z)=1, H_H(z)=z^{-1}$, $G_L(z)=z^{-1}$, $G_H(z)=1$. Its transfer function is $T(z)=z^{-1} \Leftrightarrow x^*(n)=x(n-1)$. '*n*' is the

index of incoming signals. In this system, the incoming signals are separated before processing.

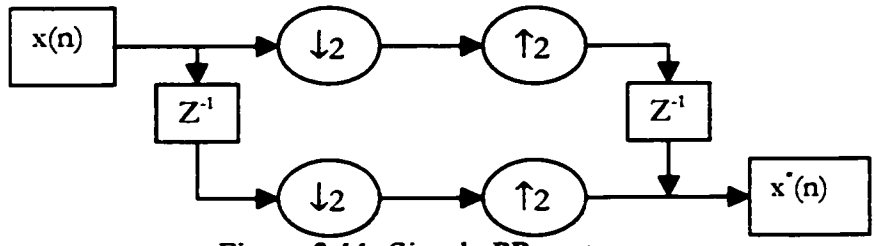


Figure 3.11: Simple PR system

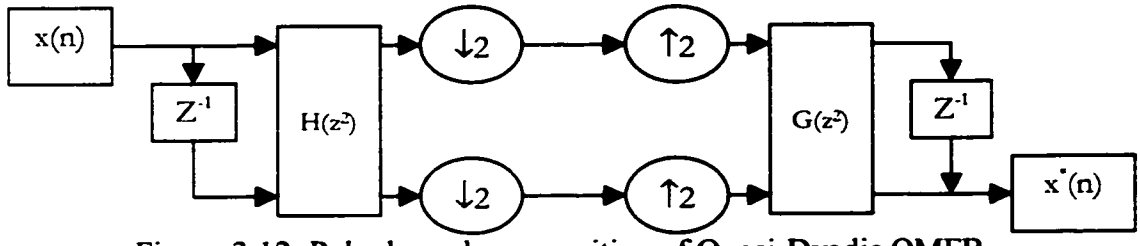


Figure 3.12: Polyphase decomposition of Quasi-Dyadic QMFB

If the Quasi-Dyadic filter is added into this system, and using polyphase decomposition, the analysis and the synthesis filters (Figure 3.12) can be written as the following matrix:

$$\begin{bmatrix} H_L(z) \\ H_H(z) \end{bmatrix} = \begin{bmatrix} H_{LE}(z^2) & H_{LO}(z^2) \\ H_{HE}(z^2) & H_{HO}(z^2) \end{bmatrix} \begin{bmatrix} 1 \\ z^{-1} \end{bmatrix} = H(z^2) \begin{bmatrix} 1 \\ z^{-1} \end{bmatrix}$$
$$\begin{bmatrix} G_L(z) \\ G_H(z) \end{bmatrix} = \begin{bmatrix} z^{-1} & 1 \end{bmatrix} \begin{bmatrix} G_{LE}(z^2) & G_{LO}(z^2) \\ G_{HE}(z^2) & G_{HO}(z^2) \end{bmatrix} = \begin{bmatrix} z^{-1} & 1 \end{bmatrix} G(z^2) \qquad (3.27)$$

Here, the second footprints stand for '***Even***' or '***Odd***' filter coefficients.

According to the Noble Identity, Figure 3.12 can be substituted by Figure 3.13 with $P(z) = G(z)H(z)$ (3.28)

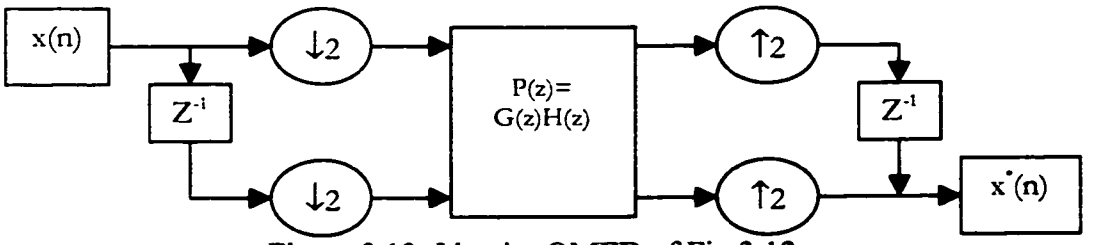


Figure 3.13: Identity QMFB of Fig 3.12

Because some filters of Quasi-Dyadic QMFB are equal, the following equations can be defined:

$$A = H_{LE}(z) = G_{LE}(z) = -H_{HE}(z) = G_{HE}(z) \tag{3.29}$$

$$B = = H_{LO}(z) = G_{LO}(z) = H_{HO}(z) = G_{HO}(z) \tag{3.30}$$

Now, *P(z)* can be expanded to:

$$\begin{aligned} P(z) = G(z)H(z) &= \begin{bmatrix} G_{LE}(z) & G_{LO}(z) \\ G_{HE}(z) & G_{HO}(z) \end{bmatrix} \begin{bmatrix} H_{LE}(z) & H_{LO}(z) \\ H_{HE}(z) & H_{HO}(z) \end{bmatrix} \\ &= \begin{bmatrix} A & B \\ -A & B \end{bmatrix} \begin{bmatrix} A & B \\ -A & B \end{bmatrix} = \begin{bmatrix} A^2 - AB & AB + B^2 \\ -A^2 - AB & -AB + B^2 \end{bmatrix} \end{aligned} \tag{3.31}$$

Since the incoming signals are separated before processing, the reconstruction signals are also separated. If *x(2n)* stands for the even incoming signals and *x(2n+1)* stands for the odd incoming signals, the above system equation can be

rewritten in the time domain: (where '*' stands for convolution and '*xr*' stands for reconstruction sample.)

$$\begin{bmatrix} xr(2n) \\ xr(2n+1) \end{bmatrix} = \begin{bmatrix} x(2n) \\ x(2n+1) \end{bmatrix} \begin{bmatrix} A(n)*A(n) - A(n)*B(n) & A(n)*B(n) + B(n)*B(n) \\ -A(n)*A(n) - A(n)*B(n) & -A(n)*B(n) + B(n)*B(n) \end{bmatrix}$$
$$= \begin{bmatrix} -2A(n)*B(n)*x(2n) \\ 2B(n)*B(n)*x(2n+1) \end{bmatrix} \tag{3.32}$$

Because of the special feature of the Quasi-Dyadic filter, any of the *B(n)* are '*0*' except the centre one. The convolution of *B(n)* by itself causes a pure delay. Moreover, the delay length is the same as the length of the Quasi-Dyadic filter. So the odd incoming signals can be perfectly reconstructed with some steps delay. The reconstruction of even incoming signals is not only dependent on *B(n)*, but also affected by *A(n)*; they may or may not be perfectly reconstructed.

One method for overcoming this drawback is inserting '*0*'s between every two input signals. Let the '*0*'s be at the even position of the incoming sequence while the odd output samples are selected only. In other words, the method changes each incoming sample to the odd position, and uses the odd output samples as reconstruction signals. Figure 3.14 illustrates that the result of this method can produce PR.

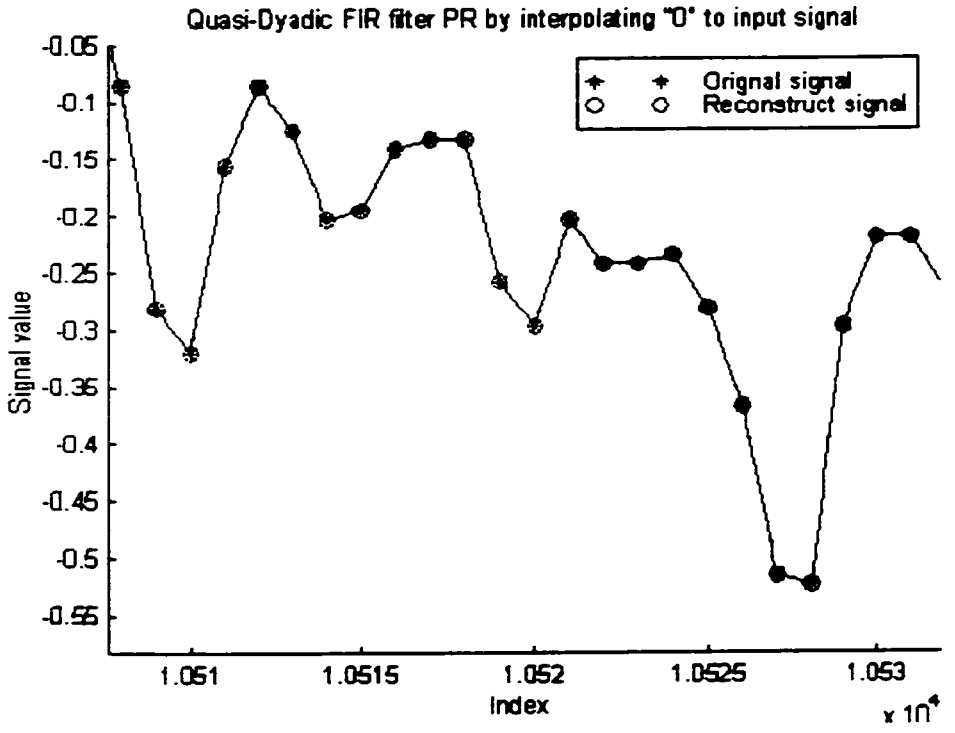


Figure 3.14: Quasi-Dyadic QMFB PR

3.3.3 Simulation of Quasi-Dyadic QMFB

The Wavelet Toolbox of Matlab (version 5.1) contains many wavelet families and it provides a flexible way to test our own wavelet.

Here we choose the Jacobian sine function *Sn(u,m)* in the frequency domain as the base lowpass filter. Based on the *Sn(u,m)*, the four FIR filters of length N are defined as in Table 3.1. Here H_L and H_H are QMFs:

$$H_H(k) = (-1)^k H_L(N-1-k) \tag{3.33}$$

Since the filter coefficients of H_H and G_H are reversed, H_H equals G_H. It is the same for H_L and G_L.

If the length of the given signal S is N, the DWT consists of log_2N stages at most. The first step (analysis level 1) produces two sets of coefficients starting from the original signal S: approximation data cA_1 and detail data cD_1. These vectors are obtained by convoluting S with the low-pass filter H_L for approximation, and with the high-pass filter H_H for detail. It is followed by dyadic decimation that just keeps the even index elements of the coefficients. If the length of each filter is equal to N, the analysis filter output signals are of length $2N-1$. After downsampling, the sample data cA_1 and cD_1 are of length $(2N-1)/2 \cong N$.

The next step (analysis level 2) splits the approximation coefficients cA_1 in two parts using the same scheme, replacing S by cA_1, and producing cA_2 and cD_2, and so on. So the wavelet decomposition of the signal S analyzed at level j has the following structure: $[cA_j, cD_j, ..., cD_1]$. (Figure 3.15)

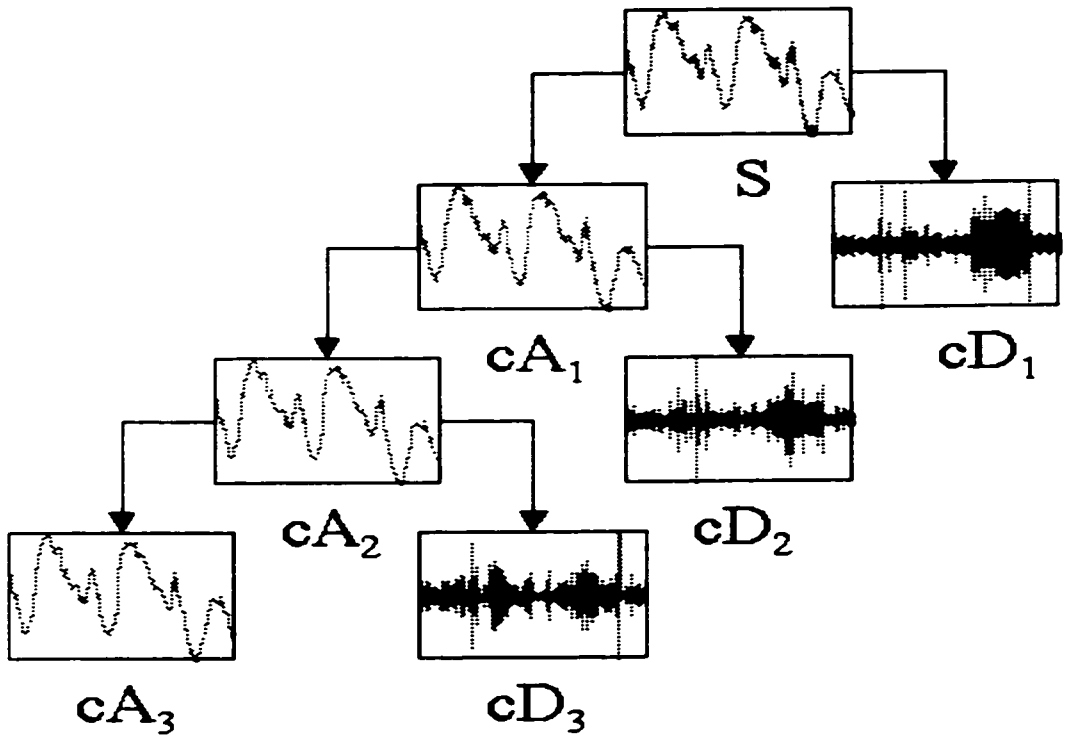


Figure 3.15: Wavelets decomposition section

Conversely, starting from cA_j and cD_j, the inverse discrete wavelet transform (IDWT) reconstructs cA_{j-1}, inverting the decomposition step by inserting zeros at odd indexed elements and convolving the results with the reconstruction filters.

This thesis just focuses on level '*1*' wavelet analysis and synthesis. The typical simulation results, which are with and without '*0*'s insertion, are shown on the following pages in Tables 3.4, 3.5, 3.6, 3.7, 3.8 and Figures 3.16, 3.17, 3.18, 3.19, 3.20.

Table 3.4: Quasi-Dyadic Fir QMFB used on music signal

Index	Original	Reconstruction	Index	Original	Reconstruction
901	-0.109375	-0.107229	902	-0.164063	-0.164063
903	-0.117188	-0.118803	904	-0.140625	-0.140625
905	-0.015625	-0.017178	906	0.085938	0.085938
907	0.156250	0.163085	908	0.218750	0.218750
909	0.218750	0.211694	910	0.101563	0.101563
911	0.031250	0.038858	912	0.015625	0.015625
913	0.054688	0.046652	914	0.132813	0.132813
915	0.179688	0.185877	916	0.226563	0.226563
917	0.195313	0.191811	918	0.164063	0.164063
919	0.062500	0.065596	920	0.007813	0.007813
921	-0.015625	-0.020245	922	0.023438	0.023438
923	0.054688	0.055414	924	0.132813	0.132813
925	0.171875	0.172010	926	0.164063	0.164063
927	0.109375	0.108997	928	0.007813	0.007812
929	-0.070313	-0.070170	930	-0.101563	-0.101563
931	-0.078125	-0.079315	932	-0.031250	-0.031250
933	0.031250	0.031735	934	0.070313	0.070313
935	0.062500	0.065205	936	0.023438	0.023438
937	-0.054688	-0.055976	938	-0.140625	-0.140625
939	-0.187500	-0.186175	940	-0.179688	-0.179688

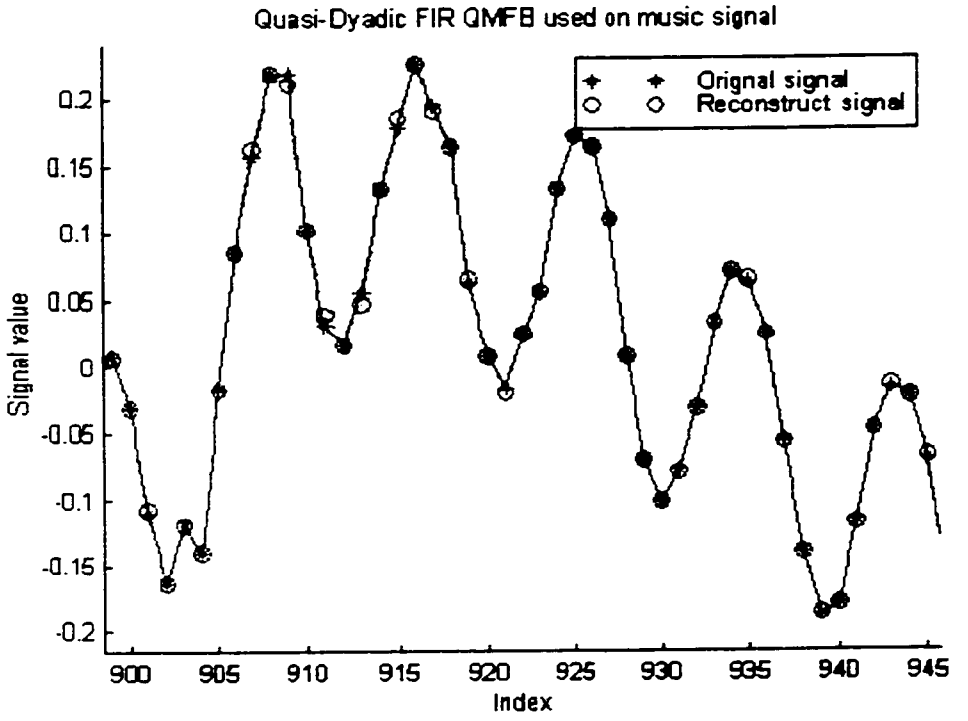


Figure 3.16: Quasi-Dyadic Fir QMFB used on music signal

Table 3.5: Quasi-Dyadic Fir QMFB used on speech signal

Index	Original	Reconstruction	Index	Original	Reconstruction
3801	0.007813	0.006586	3802	0.007813	0.007813
3803	-0.000126	0.001434	3804	-0.000126	-0.000126
3805	0.023438	0.023039	3806	0.039063	0.039063
3807	0.070313	0.073081	3808	0.078125	0.078125
3809	0.070313	0.069818	3810	0.054688	0.054688
3811	0.023438	0.026156	3812	0.007813	0.007813
3813	0.023438	0.022118	3814	0.007813	0.007812
3815	0.015625	0.017685	3816	-0.007813	-0.007812
3817	-0.031250	-0.032533	3818	-0.054688	-0.054688
3819	-0.093750	-0.092941	3820	-0.117188	-0.117188
3821	-0.125000	-0.127867	3822	-0.117188	-0.117188
3823	-0.117188	-0.115711	3824	-0.101563	-0.101563
3825	-0.085938	-0.088471	3826	-0.093750	-0.093750
3827	-0.078125	-0.076481	3828	-0.078125	-0.078125
3829	-0.039063	-0.041575	3830	-0.007813	-0.007812
3831	0.023438	0.025812	3832	0.062500	0.062500
3833	0.078125	0.076843	3834	0.062500	0.062500
3835	0.062500	0.064522	3836	0.039063	0.039063
3837	0.039063	0.038064	3838	0.046875	0.046875
3839	0.054688	0.056400	3840	0.070313	0.070313

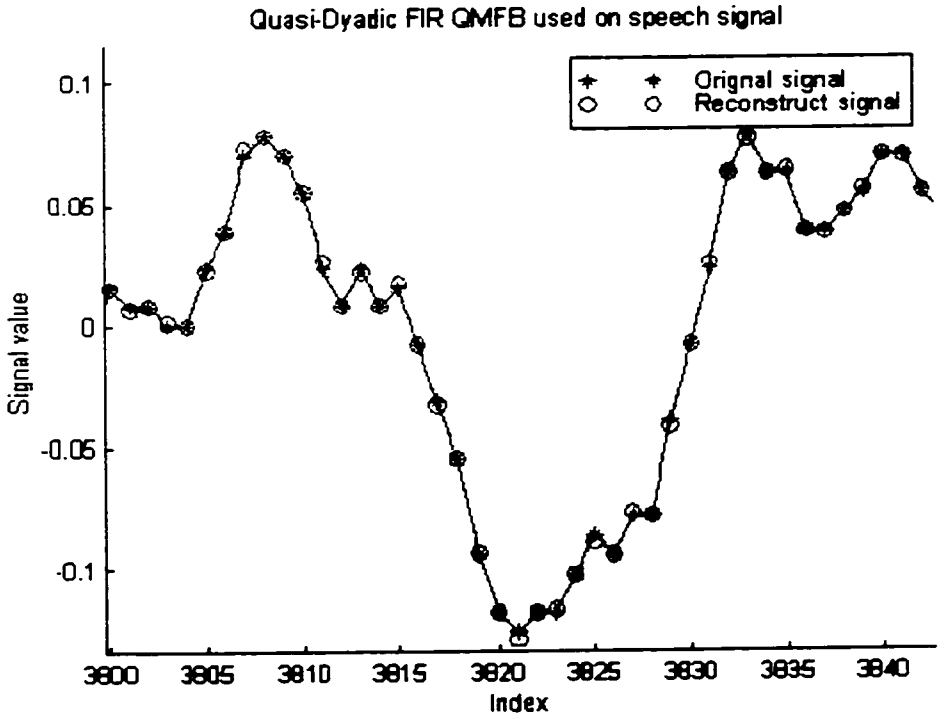


Figure 3.17: Quasi-Dyadic Fir QMFB used on speech signal

Table 3.6: Quasi-Dyadic Fir QMFB used on 1K Hz tone & silence

Index	Original	Reconstruction	Index	Original	Reconstruction
16686	0.707336	0.707336	16687	0.000366	-0.006825
16688	-0.706818	-0.706818	16689	-1.000000	-0.990679
16690	-0.707367	-0.707367	16691	-0.000366	-0.026097
16692	0.706818	0.706818	16693	0.999969	1.040746
16694	0.707336	0.707336	16695	0.000366	-0.048690
16696	-0.706818	-0.706818	16697	-0.999969	-0.936729
16698	-0.707336	-0.707336	16699	0.000000	-0.080571
16700	0.000000	0.000000	16701	0.000000	0.076851
16702	0.000000	0.000000	16703	0.000000	-0.049042
16704	0.000000	0.000000	16705	0.000000	0.027142
16706	0.000000	0.000000	16707	0.000000	-0.025720
16708	0.000000	0.000000	16709	0.000000	0.022932
16710	0.000000	0.000000	16711	0.000000	-0.007184
16712	0.000000	0.000000	16713	0.000000	0.002756
16714	0.000000	0.000000	16715	0.000000	-0.001068
16716	0.000000	0.000000	16717	0.000000	0.000387
16718	0.000000	0.000000	16719	0.000000	-0.000097
16720	0.000000	0.000000	16721	0.000000	0.000000
16722	0.000000	0.000000	16723	0.000000	0.000000
16724	0.000000	0.000000	16725	0.000000	0.000000

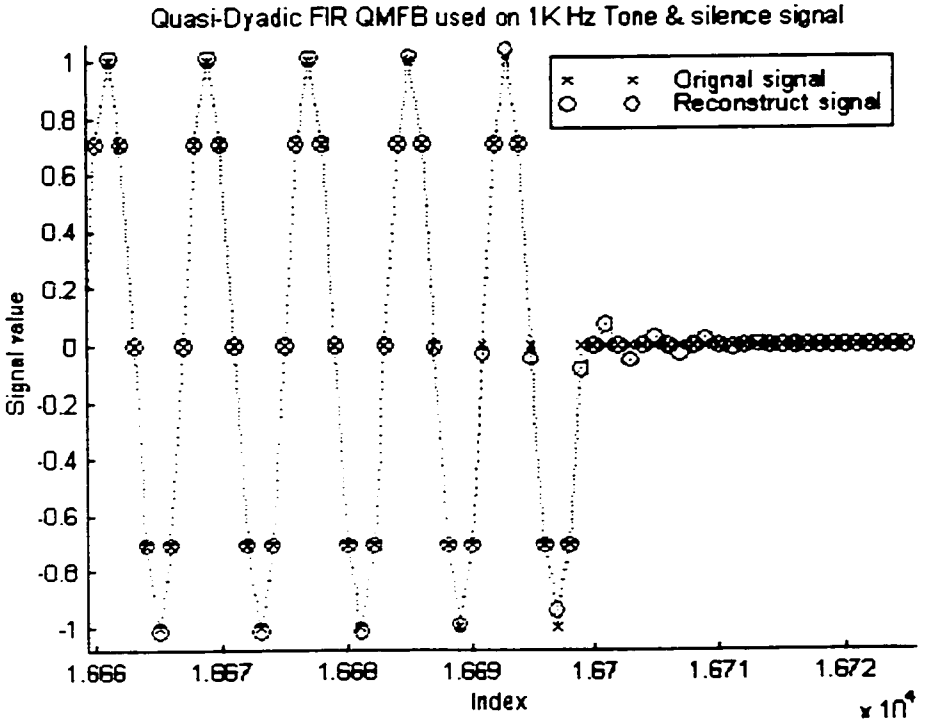


Figure 3.18: Quasi-Dyadic Fir QMFB used on 1K Hz tone & silence

Table 3.7: Quasi-Dyadic Fir QMFB with '0's insertion used on music signal

Index	Original	Reconstruction	Index	Original	Reconstruction
901	-0.109375	-0.109375	902	-0.164063	-0.164063
903	-0.117188	-0.117188	904	-0.140625	-0.140625
905	-0.015625	-0.015625	906	0.085938	0.085938
907	0.156250	0.156250	908	0.218750	0.218750
909	0.218750	0.218750	910	0.101563	0.101563
911	0.031250	0.031250	912	0.015625	0.015625
913	0.054688	0.054688	914	0.132813	0.132813
915	0.179688	0.179688	916	0.226563	0.226563
917	0.195313	0.195313	918	0.164063	0.164063
919	0.062500	0.062500	920	0.007813	0.007813
921	-0.015625	-0.015625	922	0.023438	0.023438
923	0.054688	0.054688	924	0.132813	0.132813
925	0.171875	0.171875	926	0.164063	0.164063
927	0.109375	0.109375	928	0.007813	0.007813
929	-0.070313	-0.070313	930	-0.101563	-0.101563
931	-0.078125	-0.078125	932	-0.031250	-0.031250
933	0.031250	0.031250	934	0.070313	0.070313
935	0.062500	0.062500	936	0.023438	0.023438
937	-0.054688	-0.054688	938	-0.140625	-0.140625
939	-0.187500	-0.187500	940	-0.179688	-0.179688

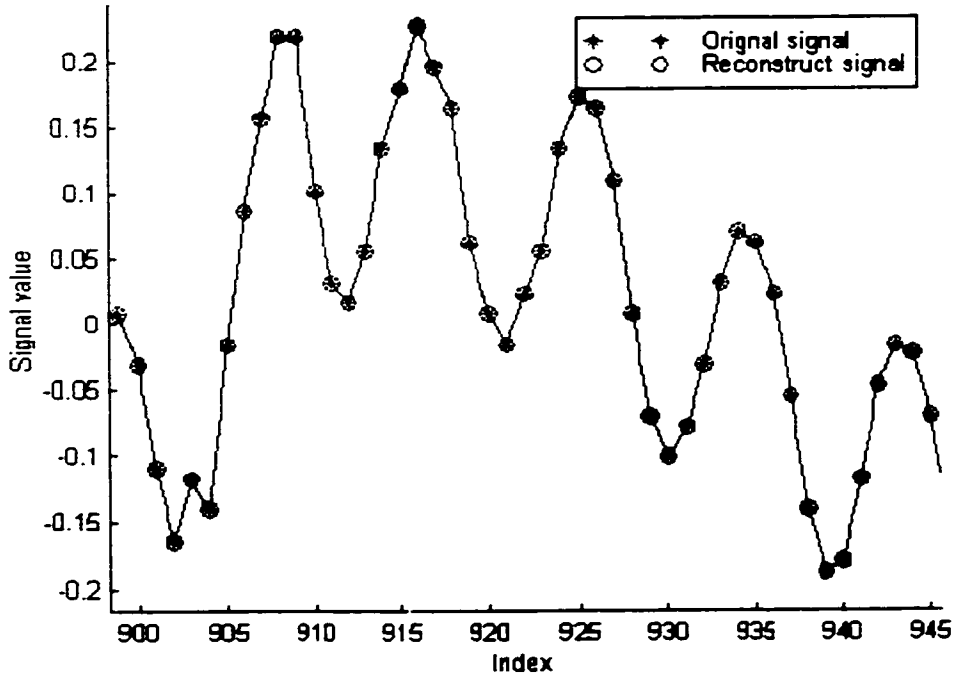


Figure 3.19: Quasi-Dyadic Fir QMFB with '0's insertion used on music signal

Table 3.8: Quasi-Dyadic Fir QMFB with '0's insertion used on 1K Hz tone & silence

Index	Original	Reconstruction	Index	Original	Reconstruction
16686	0.707336	0.707336	16687	0.000366	0.000366
16688	-0.706818	-0.706818	16689	-1.000000	-1.000000
16690	-0.707367	-0.707367	16691	-0.000366	-0.000366
16692	0.706818	0.706818	16693	0.999969	0.999969
16694	0.707336	0.707336	16695	0.000366	0.000366
16696	-0.706818	-0.706818	16697	-0.999969	-0.999969
16698	-0.707336	-0.707336	16699	0.000000	0.000000
16700	0.000000	0.000000	16701	0.000000	0.000000
16702	0.000000	0.000000	16703	0.000000	0.000000
16704	0.000000	0.000000	16705	0.000000	0.000000
16706	0.000000	0.000000	16707	0.000000	0.000000
16708	0.000000	0.000000	16709	0.000000	0.000000
16710	0.000000	0.000000	16711	0.000000	0.000000
16712	0.000000	0.000000	16713	0.000000	0.000000
16714	0.000000	0.000000	16715	0.000000	0.000000
16716	0.000000	0.000000	16717	0.000000	0.000000
16718	0.000000	0.000000	16719	0.000000	0.000000
16720	0.000000	0.000000	16721	0.000000	0.000000
16722	0.000000	0.000000	16723	0.000000	0.000000
16724	0.000000	0.000000	16725	0.000000	0.000000

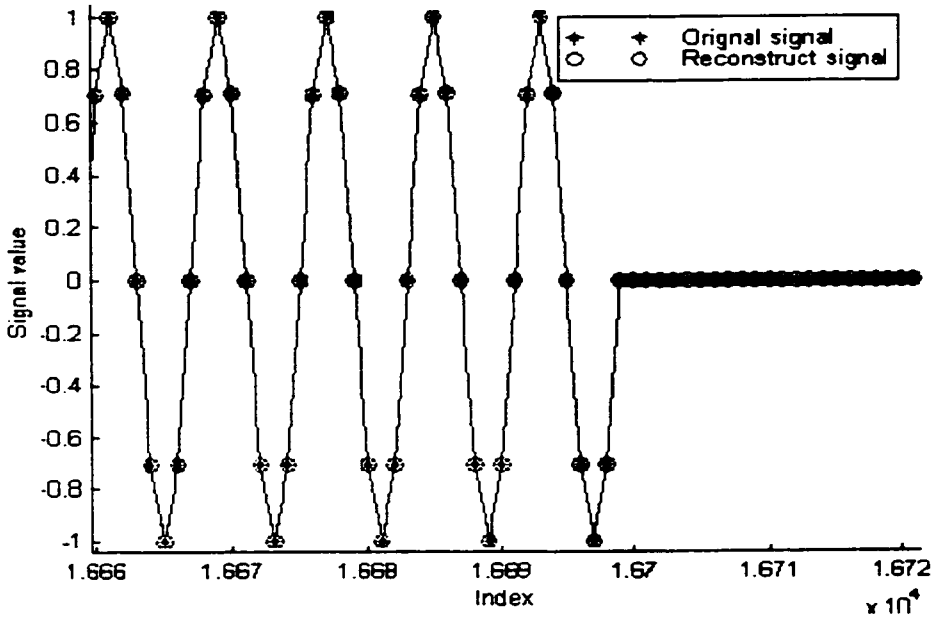


Figure 3.20: Quasi-Dyadic Fir QMFB with '0's insertion used on 1K Hz tone & silence

3.4 Implementation

The outstanding feature of a Quasi-Dyadic filter is that most of its odd coefficients are dyadic data. A binary coefficient or dyadic coefficient is an integer divided by a power of 2. Multiplication by a binary number can be executed entirely by 'shifts' and 'adds'. Round-off error is eliminated. In addition, the filter needs less time and less space. The TMS320C6201 evaluation module (EVM) system is selected for implementing the Quasi-Dyadic filters in a real system.

3.4.1 Implementation system

The TMS320C6x EVM is a low-cost, general-purpose platform for the development, analysis, and testing of 'C6x digital signal processor (DSP) algorithms and applications. The 'C6x EVM allows us to evaluate the 'C6x DSP and algorithms to determine if they meet our application requirements. The 'C6x EVM hardware design information and software application programming interfaces (APIs) also provide a reference design that facilitates users to develop 'C6x-based hardware and software. The TMS320C6x EVM hardware functional block overview is shown in Figure 3.21. [8]

The development software, Code Composer, provides coding and real-time testing environments for users. These include the tools for configuring, building, debugging, tracing, and analyzing programs. The DSP/BIOS foundation software, included in Code Composer Studio, furnishes a small

firmware kernel with basic run-time services that software developers can embed on target DSP hardware. The embedded DSP/BIOS run-time library and DSP/BIOS plug-ins support a new generation of testing and diagnostic tools that allow developers and integrators to probe, trace, and monitor a DSP application during its course of execution. The system running status may be viewed via this real-time monitoring.

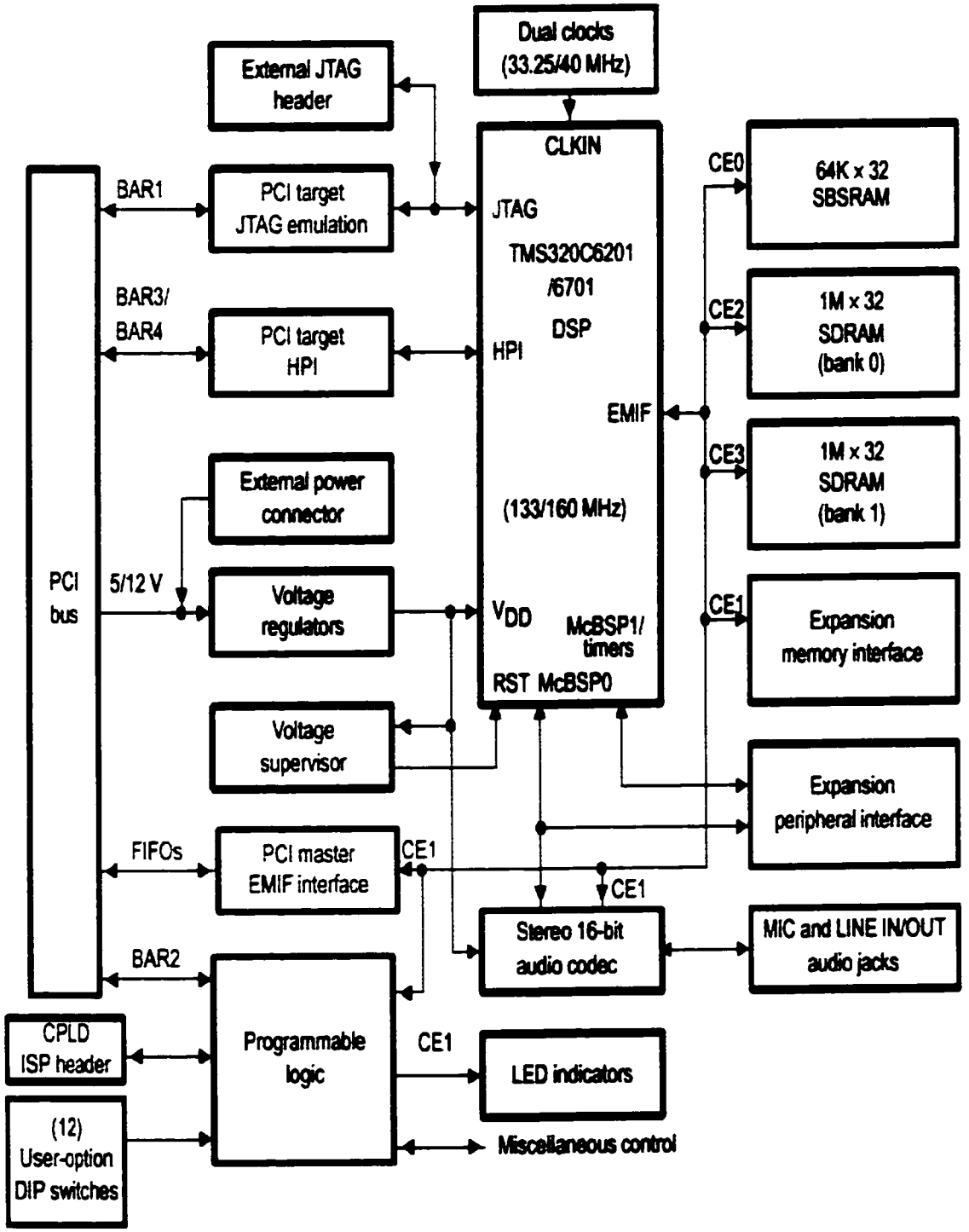


Figure 3.21: TMS320C6201 EVM functional overview

3.4.2 Quasi-Dyadic QMFB Implementation

The kennel part of Quasi-Dyadic QMFB implementation is shown in Figure 3.22. A 1K Hertz tone is generated from the hardware; then this tone goes through the Quasi-Dyadic QMFB and the reconstruction signals are produced. The simulation code uses the onboard CODEC to play the original and reconstructed sound. Since the TMS320c6201 is a fixed-point DSP, the coefficients must be normalized based on 0xffff (32767) ---- full value of 16 bits.

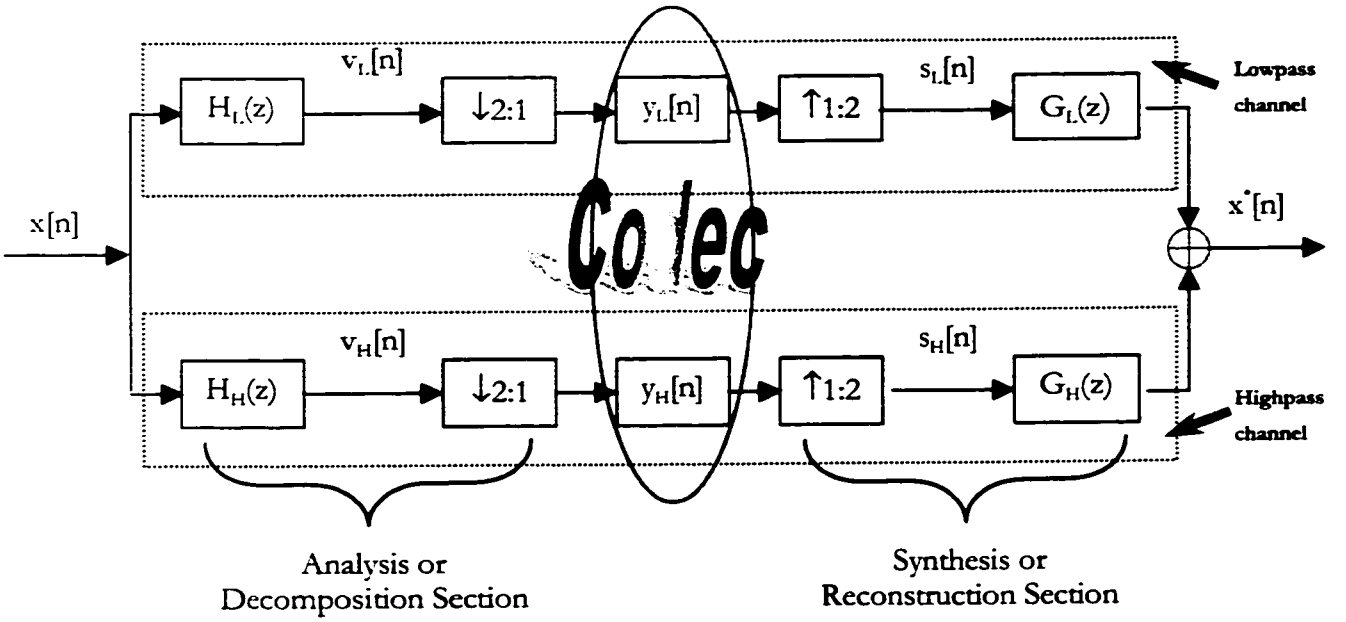


Figure 3.22: Quasi-Dyadic QMFB

The FIR filter is calculated using the following equation:

$$y(n)=\sum_{k=0}^{N-1}h(k)x(n-k) \tag{4.1}$$

Here, '*y(n)*' is the output signal, '*x(n-k)*' is the input signal, '*h(k)*' represents the filter coefficients, '*n*' is the index, and '*N*' is the filter order. Since most of the non-zero coefficients are in powers of two (2^{-1}, 2^{-2}, 2^{-3}...), the multiply operation formula (4.1) may be changed to a shift operation. In TMS320C6201, the general shift type instructions only need one cycle to be executed. However, the multiply instructions need four cycles. Therefore, the filter processing will be more efficient after the carefully adjusted calculation procedure.

Chapter 4

CONCLUSIONS AND SUGGESTIONS FOR FUTURE STUDY

The design procedure for a new FIR filter, the Quasi-Dyadic filter is provided. Since most of the coefficients of this Quasi-Dyadic filter are of the form $\alpha 2^{-n}$ (where α is the scale of the coefficients and n is a positive integer), many of the 'multiply' operations may be changed to simple 'shift' operations in filtering calculations. This change saves instruction cycles in real-time filter systems. It also improves the accuracy problem in general.

It has been shown that the QMFB using the Quasi-Dyadic filter provides good results. Although the Quasi-Dyadic QMFB-reconstructed signals exhibit minor distortion for real-world audio signals, the reconstructed sound is virtually the same, and hence quite practical in many applications, (e.g., data compression, pattern recognition, etc.). Nevertheless, this thesis also presents a solution to provide perfect reconstruction by interleaving padded ' *0*'s ' to the incoming signal. The latter method can be used when perfect reconstruction is required. In addition, implementation of these new methods has been successfully performed on a commercial DSP board.

Future work outside the scope of this thesis may involve using the Quasi-Dyadic QMFB in two-dimensional applications, such as image processing. Reducing the error rate of the reconstruction may be a direction for future research. Study of possible fully dyadic coefficient designs is also suggested.

REFERENCES:

1. T. D. Tran, "The BinDCT: fast multiplierless approximation of the DCT", (IEEE Signal Processing Letters, to appear, Jun. 2000), World Wide Web: http://thanglong.ece.jhu.edu/Tran/Pub/intDCT_SPL.pdf, May 2000.
2. T. D. Tran, R. L. de Queiroz, and T. Q. Nguyen, "Linear phase perfect reconstruction filter bank: lattice structure, design, and application in image coding", IEEE Trans. on Signal Processing, vol. 48, pp. 133-147, Jan. 2000.
3. Dr. James J. Soltis, personal communication, Jan. 2000
4. T. D. Tran, "M-channel linear phase perfect reconstruction filter bank with integer coefficients", SPIE International Symposium on Optical Science, Engineering, and Instrumentation, Session: Wavelet Applications in Signal and Image Processing. Denver, July 1999.
5. Ashok Ambardar, "Analog and Digital Signal Processing" (2nd edition), Nelson, 1999
6. T. D. Tran, "Fast multiplierless approximation of the DCT", 33rd Annual Conference on Information Sciences and Systems, Baltimore, MD, March 1999.
7. Wijay K. Madsetti, Douglas B. Williams, "The Digital Signal Processing Handbook", CRC Press/IEEE Press, 1998
8. Texas Instruments Inc. "TMS320C6201/6701 Evaluation Module Technical Reference", December 1998
9. Fengshi Chen, "Wavelet Transform in Signal Processing Theory and Applications", National Industry, 1998 (In Chinese)

10. T. D. Tran and T. Q. Nguyen, "On M-channel linear phase FIR filter banks and application in image compression", IEEE Trans. on Signal Processing, vol. 45, pp. 2175-2187, Sept. 1997.
11. Dong Wei, Brian L. Evans, and Alan C. Bovik, "Loss of Perfect Reconstruction in Multidimensional Filter banks and Wavelets Designed via Extended McClellan Transformations", IEEE Signal Processing Letters, vol. 4, no. 10, pp. 295-297, Oct. 1997
12. Shongnian Zhao, "Wavelet Transform and Wavelet Analysis", Electronics Industry, 1997 (In Chinese)
13. Shongnian Zhao, "Wavelet Transform and Wavelet Analysis", Electronics Industry, 1997 (In Chinese)
14. Jianpin Li, "Wavelet Analysis and Signal Processing", Chongqing, 1997 (In Chinese)
15. Michel Misiti, "Wavelet Toolbox User's Guide" (version 1), Mathworks, 1997
16. Weizheng Ma, "Wavelet Analysis and Wavelet Transform", Huanan University, 1996 (In Chinese)
17. John Woods, "Subband Coding", Kluwer Academic Press, 1990
18. Andreas Antonous, "Digital Filters Analysis Design and Applications" (2nd edition), McGraw-Hill, 1993
19. J.F. Kaiser, "Digital Filters", NY, Wiley, 1966
20. Abramowitz, M and I.A. Stegun, "Handbook of Mathematical Functions", Dover Publications, 1965: section 17.6
21. Mallat S., "A Theory for Multi-Resolution Decomposition the Wavelet Representation", IEEE Trans Pattern Anal Machine Intel 1989 11(7): page 674-693
22. P.P. Vaidyanathan, "Quadrature Mirror Filter Banks, M-Band Extensions and Perfect-Reconstruction Techniques", IEEE ASSP, 4-20, July 1987

23. J. D. Johnston. "A Filter Family Designed using Quadrature Mirror Filter Banks", Proc. ICASSP, 1980, page: 291-294
24. Martin Vetterli, " A Theory of Multirate Filter Banks", IEEE Trans. ASSP-35 (3), March 1987, page: 356-372.
25. Vijay K. Jain, "Quadrature Mirror Filter Design in the Time Domain", IEEE Trans. ASSP-32 (2), April 1984, page: 353-360.
26. Truong Q. Nguyen, "Two-Channel Perfect-Reconstruction FIR QMF Structures which Yield Linear-Phase Analysis and Synthesis Filters", IEEE Trans. ASSP-37 (5), May 1989, page: 676-690.
27. Truong Q. Nguyen, "Structures for M-Channel Perfect-Reconstruction FIR QMF Banks which Yield Linear-Phase Analysis Filters", IEEE Trans. ASSP-38 (3), March 1990, page: 433-446.
28. Ramin Safari, "Linear Phase Lowpass FIR Filter Design using the Jacobian Elliptic Sine", course project report, Jan. 1995.
29. P.P. Vaidyanathan, "Multirate Systems and Filter Banks", Prentice Hall, 1993
30. Rioul O, Flandrin P., "Time-Scale Energy Distributions: A General Class Extending Wavelet Transforms", IEEE Trans. Sig. Proc., July 1992
31. Zou H, Tewfik A H, Xu W. "Completeness and Stability of Partial Dyadic Wavelet Domain Signal Representations", Proc. ICASSP, 1993
32. Jorgensen P., "Choosing Discrete Orthogonal Wavelets for Signal Analysis and Approximation", Proc. ICASSP, 1993
33. Vaidyanathan P. P., "Theory and Design of M-channel Maximally Decimated Quadrature Mirror Filters with Arbitrary M, having Perfect Reconstruction Property", IEEE Trans. ASSP-35, April 1987
34. Vaidyanathan P. P., "Multirate Digital Filters, Filter Banks, Polyphase Networks, and Applications: A Tutorial", Proc. of IEEE, vol. 78, No. 1, Jan. 1990

35. Soman A. K., Vaidyanathan P. P., "Wavelet and Paraunitary Filter Banks", IEEE Trans. Signal Processing, vol. 41,No. 3, page 1170-1183, March 1993
36. I. Daubechies, "Ten Lectures on Wavelets, CBMS-NSF Series on Applied Mathematics", SIAM, 1992.
37. I. Daubechies, "Orthonormal bases of compactly supported wavelets", Communication on Pure and Applied Mathematics, Vol. XLI, pp. 909-996, 1988.
38. M. Vetterli and C. Herley, "Wavelets and filter banks", IEEE Trans. on S P, vol. SP-40, 1992.
39. Z. Doganata, P. P. Vaidyanathan and T. Q. Nguyen, "General Synthesis Procedures for FIR Lossless Transfer Matrices for Perfect Reconstruction Multirate Filter Bank Application", IEEE Trans. on Acoustics, Speech and Signal Processing, pp. 1561-1574, Oct. 1988.
40. P. P. Vaidyanathan and V. C. Liu, "Classical Sampling Theorems in the Context of Multirate and Polyphase Digital Filter Bank Structures", IEEE Trans. on ASSP, September 1988, pp. 1480-1495.
41. C. D. Creusere and S. K. Mitra., "A simple methor for designing high-quality prototype filters for m-band pseudo QMF banks", IEEE Trans. on Signal Processing, 43(4):1005--1007, April 1995.
42. M. H. Er and C. K. Siew, "Design of FIR filters using quadrature programming approach", IEEE Trans. on Circuits and Systems-II, 42(3): 217--220, March 1995.
43. L. M. Milne-Thomson, "Jacobian Elliptic Function Tables", NY Dover, 1950

APPENDIX: SIMULATION & IMPLEMENTATION CODE

1. Quasi-Dyadic coefficient generation and analysis program: qdga.m

```
%File Name: qdga.m
%This program is used to generate Quasi-Dyadic filter coefficients
%and analysis them.

clear all; close all; format long G;

sec_len=512;                                          % Incoming data section length
nq=round(sec_len/2); h_nq=nq/2;                       % Nyquist frequency & its half

%k=0.9999947610549319;                                % best k in Matlab
%m=k^2;                                               % 'm' is the parameter of Jacobian function modulus squared
m=0.99998952213731035;                                % best m in Matlab

K=ellipke(m);                                         % Get the elliptic first integrals 'K'
u=K:K/h_nq:(3*K);%-K/h_nq);   % Generate a serials of K for Jacobian elliptic function coefficient u
%u=-K:K/h_nq:K-K/h_nq;        % Generate a serials of K for Jacobian elliptic function coefficient u
%sn=0.5*(ellipj(u,m)+1);                              % Here is shifted sn function in frequency domain

K=ellipke(m);                                         % Get the elliptic first integrals 'K'
KP=ellipke(1-m);                                      % KP
rt=-pi*KP/K;
%rt=-0.69314718055995;
idx=1;                                                % index of sn
md_sn=0;                                              % md_sn is no-scale sn
for idx=1:length(u)                                   % idx is the index of sn
   tp_sn=0;
   uu=u(idx);
   for n=0:2^5;                                       % iterates n times, Matlab biggest n=2^10 not overflow
      alf=exp((n+0.5)*rt);
      beta=1-exp((2*n+1)*rt);
      tp=pi*uu/(2*K);
      tp_sin=sin((2*n+1)*tp);
      tp_sn=tp_sn+alf/beta*tp_sin;
   end
   md_sn(idx)=tp_sn;
end
sn=md_sn*2*pi/(K*m^0.5);   % sn
%*************************************************************************
ss_sn=0.5*(sn+1);                                     % ss_sn is shift and scaled sn
%*************************************************************************
ss_sn_add0=ss_sn;
ss_sn_add0(nq+2:2*nq+2)=ss_sn_add0(nq+1:-1:1);
nf_idx=(0:sec_len-1)./(sec_len-1);
```

```
plot(nf_idx, ss_sn_add0(1:512),'r:'); hold on;
plot(nf_idx(1:256), ss_sn_add0(1:256));
title('Scaled & shift sn(u,m) with magic k=.99999476');
ylabel('sn(u,m)'); xlabel('Normalized frequency');
%plot(ss_sn, 'r*');
%plot(u, 'g');
legend('Expand part shape', 'Lowpass filter shape');

% Caculate fir filter coefficents here
hc=ifft(ss_sn_add0); h=real(hc);                                    % Back to time domain
%hc=ifft(ss_sn); h=abs(hc);                                         % Back to time domain

figure(2); hold on;                                                 % Test coefficients is 2's power
len_k=length(h)/4; test_step=1:2:len_k;
%h_odd=abs(2*h(test_step)./h(1)); h_even=abs(2*h(test_step+1)./h(1));
h_odd=real(h(test_step)); h_even=real(h(test_step+1));
len_power=len_k/2; power_step=1:len_power;
fig_odd=h_odd.*2.^power_step; fig_even=h_even.*2.^power_step;
co_len=length(fig_odd)+length(fig_even);
co_fig=ones(1,co_len);
co_fig(1:2:co_len)=fig_odd;
co_fig(2:2:co_len)=fig_even;
stem(co_fig(1:64));
title('Relationship between Quasi-Dyadic filter coefficients and powers of 2');
xlabel('Index'); ylabel('Coeficients * 2^n n=index/2');
%stem(fig_odd(1:36), 'b'); stem(fig_even(1:36), 'r');
%legend('odd', 'even');

figure(3); hold on;
N=23;
for ii=1:(N-1)/2+1
  g(ii)=h((N-1)/2+2-ii);
end
for ii=1:(N-1)/2
  g(N+1-ii)=g(ii);
end
plot(g,'*');
title('Quasi-Dyadic filter coefficients');
xlabel('Index'); ylabel('Coeficients value');

f=abs(fft(g));
figure(4); hold on;
fqmf=abs(fft(qmf(g)));
q_idx=(0:22)./22;
plot(q_idx,f); plot(q_idx,fqmf,'r:');
title('Two-channel Quasi-Dyadic QMF bank, order=23');
xlabel('Normalized frequency'); ylabel('Magnitude');
```

2. Simulation using music, speech or tone: **qdsimu.m**

```
% File Name: qdsimu.m
% This file is used to simulate the signals using Quasi-Dyadic Filter bank.
% This file is also check the results of the reconstruction signals.
% g is the 23 order of Quasi-Dyadic Filter coefficients
% hb_qmf is the highpass decomposition filter
% gb is the lowpass reconstruction filter
% gb_qmf is the highpass reconstruction filter
% x is the original signal
% xe is the reconstruction signal

clear all; close all;
format long g;
hold on;

g=[-0.00492036427341 0  0.00983630456385 0 -0.01974931798687 0 ...
   0.04037500055841 0 -0.08925218167063 0  0.31206897810975 0.5 ...
   0.31206897810975 0 -0.08925218167063 0  0.04037500055841 0 ...
  -0.01974931798687 0  0.00983630456385 0 -0.00492036427341];

load music;y=music;
%load speech;y=speech;

mean_y=mean(y);                          % prepare none '0' input signal to prevent 'divided by zero'
for ii=1:length(y)
   if y(ii)==0
      x(ii)=mean_y;
   else
      x(ii)=y(ii);
   end;
end;

%x=(rand(1,40)-0.5)*20;y=x;

x_len=length(x)
%sound(x, 8000);                                          % Listen the sound record
xd=x;                                          % Make highpass and lowpass input signal as equal
hb=g;
hb_len=length(hb);
hb_qmf=qmf(hb,1);                                        % QMF filter coefficients design

aa=[];                                                   % Prepare denominator of the filter
aa(1)=1;
yl=filter(hb,aa,x);                                      % lowpass analysis Section
ylc=yl;                                                  % Do down-sample & up-sample.
ylc(2:2:x_len)=0;                                        % Change filtered samples to '0' in alternation

yh=filter(hb_qmf,aa,xd);                                 % Highpass analysis Section
yhc=yh;
yhc(2:2:x_len)=0;

%gb=hb(length(hb):-1:1);
```

```
gb=wrev(hb);                                          % Reverse filter coefficient
gb_qmf=wrev(hb_qmf);

xl=filter(gb,aa,ylc);                                 % Lowpass and highpass Synthesis Sections
xh=filter(gb_qmf,aa,yhc);
%xl=conv(ylc, gb);
%xh=conv(yhc, gb_qmf);

xe=(xl+xh)*2;                                         % Synthesis the reconstruction signal and scale it

hold on;
plot(x(1:x_len-(hb_len-1)),'mx');
plot(xe(hb_len:x_len),'ro');
plot(x(1:x_len-(hb_len-1)),'b');
plot(xe(hb_len:x_len),'r:');
ylabel('Signal value'); xlabel('Index');
legend('Orignal signal', 'Reconstruct signal');
%title('Quasi-Dyadic FIR QMFB used on random integer signal');
title('Quasi-Dyadic FIR QMFB used on music signal');
%title('Quasi-Dyadic FIR QMFB used on speech signal');

% The following parts are error comparing and plotting

err=abs(x(1:x_len-(hb_len-1))-xe(hb_len:x_len));      % Shift reconstruction signal
maxerr=max(err)
maxerreven=max(err(1:2:length(err)))                  % Matlab matrix index from '1' not '0'
maxerrodd=max(err(2:2:length(err)))

errptg=abs(err(1:x_len-(hb_len-1))./x(1:x_len-(hb_len-1)))*100;
maxerrptg=max(errptg)
meanerrptg=mean(errptg)
```

3. Adjusted Quasi-dyadic Filter (Perfect Reconstruction) **prqd.m**

```
% File Name: prdq.m
% This file is used to do perfect reconstruction using Quasi-Dyadic Filter
% bank. It inserts '0' into the incoming signals.

% g is the 23 order of Quasi-Dyadic Filter coefficients
% hb is the lowpass decomposition filter
% hb_qmf is the highpass decomposition filter
% gb is the lowpass reconstruction filter
% gb_qmf is the highpass reconstruction filter
% x is the original signal
% xe is the reconstruction signal

clear all; close all;
format short;
```

```
hold on;

g=[-0.00492036427341 0  0.00983630456385 0 -0.01974931798687 0 ...
   0.04037500055841 0 -0.08925218167063 0  0.31206897810975 0.5 ...
   0.31206897810975 0 -0.08925218167063 0  0.04037500055841 0 ...
  -0.01974931798687 0  0.00983630456385 0 -0.00492036427341];


%x=(rand(1,64)-0.5)*20;

load music;                                  % prepare none '0' input signal to prevent 'divided by zero'
y=music;

mean_y=mean(y);
for ii=1:length(y)
   if y(ii)==0
      x(ii)=mean_y;
   else
      x(ii)=y(ii);
   end;
end;

x_len=length(x)
%sound(x, 8000);                                          % Listen the sound record

xi_len=2*x_len;
xi=zeros(1, xi_len);
xi(2:2:xi_len)=x;

%xd=x;                                       % Make highpass and lowpass input signal as equal
hb=g;
hb_len=length(hb);
hb_qmf=qmf(hb,1);                                        % QMF filter coefficients design

aa=[];                                                   % Prepare denominator of the filter
aa(1)=1;

yl=filter(hb,aa,xi);                                                % lowpass analysis Section
ylc=yl;                                                  % Do down-sample & up-sample.
ylc(2:2:xi_len)=0;                                % Change filtered samples to '0' in alternation

yh=filter(hb_qmf,aa,xi);                                            % Highpass analysis Section
yhc=yh;
yhc(2:2:xi_len)=0;

%gb=hb(length(hb):-1:1);
gb=wrev(hb);                                                        % Reverse filter coefficient
gb_qmf=wrev(hb_qmf);

xl=filter(gb,aa,ylc);                                % Lowpass and highpass Synthesis Sections
xh=filter(gb_qmf,aa,yhc);
```

```
xie=(xl+xh)*2;
%xe=(xl+xh)*2;                          % Synthesis the reconstruction signal and scale it

xe=xie(2:2:length(xie));
err=x(1:x_len-(hb_len/2-1))-xe(hb_len/2:x_len);
plot(err,'g*'); figure;
%plot(x(1:x_len-(hb_len/2-1))-xe(hb_len/2:x_len),'g*'); figure;
hold on;
plot(x,'mx');
plot(xe(hb_len/2:x_len),'ro');
plot(x);
title('Quasi-Dyadic FIR filter PR by inserting "0" to input signal');
ylabel('Signal value'); xlabel('Index');
legend('Orignal signal', 'Reconstruct signal');

% The following parts are error comparing and plotting
maxerr=max(abs(err))
```

4. Simulation code on TI EVM

qdsimu.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <common.h>
#include <mcbspdrv.h>
#include <mcbsp.h>                                  /* mcbsp devlib */
#include <board.h>
#include <codec.h>
#include <mathf.h>

#define TONE_PB_SAMPLE_RATE     8000                /* Hz */
#define TONE_FREQ               1000                /* Hz */
#define TONE_SCALE              32767
#define CONT_TONE_PB_SECS       1                   /* Seconds */

int qmf(short *h, unsigned short h_size)
{
  unsigned short ii;

  for (ii=1; ii<=h_size; ii++)
    if (ii%2 == 1) h[ii-1] *= -1;

  return(0);
}

int filter(unsigned int x_len, short *x, short *y, unsigned short h_len,
        short *h, short flag)
{
```

```
unsigned int ii, jj, sum, index;

  ii=0;                                                    /* just for testing */
  for (jj=1; jj<=x_len; jj++)
  {
    sum=0;
    for (ii=1; ii<=h_len; ii++)
    {
      index = ii + jj;
      sum = sum + x[index] * h[ii];
    }
    if (flag == 1) y[jj] += sum >> 15;
    else y[jj] = sum >> 15;
  }
  ii=0;                                                    /* Just for testing */
  return(0);
}

int filter2(unsigned int x_len, short *x, short *y, unsigned short h_len,
        short *h, short flag)
{
  unsigned int ii, jj, sum, index;

  ii=0;                                                    /* just for testing */
  for (jj=1; jj<=x_len; jj++)
  {
    sum=0;
    for (ii=1; ii<=h_len; ii++)
    {
      index = ii + jj;
      if (ii<=10)
      {
      if (h[ii]==-1) sum = sum - x[index]>>((h_len+1)/2-ii);
      if (h[ii]== 1) sum = sum + x[index]>>((h_len+1)/2-ii);
      }

      if (ii>=14)
      {
      if (h[ii]==-1) sum = sum - x[index]>>(ii-(h_len+1)/2);
      if (h[ii]== 1) sum = sum + x[index]>>(ii-(h_len+1)/2);
      }
      else
          if (ii==(h_len+1)/2) sum = sum + x[index] + x[index]>>1;
          else sum = sum + x[index];
    }
    sum *= 10382;                                          /* 32767 */
    if (flag == 1) y[jj] += sum >> 15;
    else y[jj] = sum >> 15;
  }
  ii=0;                                                    /* Just for testing */
```

```
  return(0);
}

static void led_blink( int count, int ms_period )
{
  int   i;
  for (i=0;i<count;i++)
  {
    evm_led_enable(0); evm_led_enable(1); delay_msec(ms_period/2);
    evm_led_disable(0); evm_led_disable(1); delay_msec(ms_period/2);
  }
}

/*****************************************************************************/
/* GenTone() - Generate tone samples with indicated frequency.               */
/* Reference code: TI examples                                               */
/*****************************************************************************/
int GenTone(                                                    /* RET: OK or ERROR */
        short       *pBuffer,                                   /* I/O: buffer for generated data */
        int         buffSize,                                   /* IN: number of shorts in buffer */
        float       toneFreq,                                   /* IN: requested tone frequency (Hz) */
        float       sampleRate,                                 /* IN: current CODEC sample rate (Hz) */
        int         leftRightSel)                               /* IN: LEFT, RIGHT, or BOTH */
{
 int i;
 float omega;
 float deltaT;
 short scaledVal;
 unsigned int scl=TONE_SCALE;                                   /* 32767 */

 memset((unsigned char *)pBuffer,0,(buffSize * sizeof(short)));

 for (i=0; i< buffSize; i++)
 {
  omega= 2 * PI * toneFreq;
  deltaT= (i/2) / sampleRate;

  scaledVal= (short)(scl * (sinf(omega*deltaT)));

  if ( (leftRightSel == RIGHT) || (leftRightSel == BOTH) )
    pBuffer[i] = scaledVal;
  i++;
  if ( (leftRightSel == LEFT) || (leftRightSel == BOTH) )
    pBuffer[i]  = scaledVal;
 }

 return(0);
}

/*****************************************************************************/
/* CodecExContTonePb() - Play continuous tone */
```

```
/* continuous (dma autoinitialization) mode. */
/*****************************************************************************/
int CodecExContTonePb( Mcbsp_dev dev, int leftRightSel, int duration )
{

 Mcbsp_config  mcbspConfig;
 unsigned int  numBytes;
 int           sampleRate;
 short         *pBuffer;

 short h[]={-1, 0, 1, 0, -1, 0, 1, 0, -1, 0, 1, 8/5, 1, 0, -1, 0, 1, 0, -1, 0, 1, 0, -1};

 short         *pReBuffer;
 short         *tpBuffer;
 unsigned int RenumBytes;
 unsigned int  ii;
 unsigned int  h_bytes;

 sampleRate = TONE_PB_SAMPLE_RATE;

 /*****************************************************************************/
 /* allocate buffer for audio data                                            */
 /*****************************************************************************/
 numBytes = TONE_PB_SAMPLE_RATE * sizeof(int);
 pBuffer = (short *)(malloc( numBytes ));
 if (pBuffer == NULL)
 {
  DEBUG("Error allocating capture buffer 1 in codecExContTonePb()");
  return(ERROR);
 }

 h_bytes = sizeof(h);
 RenumBytes = numBytes + h_bytes;
 pReBuffer = (short *)(malloc( RenumBytes ));
 tpBuffer = (short *)(malloc( RenumBytes ));

 if (pReBuffer == NULL || tpBuffer == NULL)
 {
  DEBUG("Error allocating capture buffer 2 in codecExContTonePb()");
  return(ERROR);
 }

 /*****************************************************************************/
 /* Synthesize tone                                                           */
 /*****************************************************************************/

 GenTone(pBuffer,numBytes/2,(int)TONE_FREQ,(int)sampleRate,leftRightSel);

 /*****************************************************************************/
 /* configure mcbsp transmitter                                               */
 /*****************************************************************************/
```

```
memset(&mcbspConfig,0,sizeof(mcbspConfig));

mcbspConfig.loopback            = FALSE;

mcbspConfig.tx.update           = TRUE;
mcbspConfig.tx.clock_polarity   = CLKX_POL_RISING;
mcbspConfig.tx.frame_sync_polarity= FSYNC_POL_HIGH;
mcbspConfig.tx.clock_mode       = CLK_MODE_EXT;
mcbspConfig.tx.frame_sync_mode  = FSYNC_MODE_EXT;
mcbspConfig.tx.phase_mode       = SINGLE_PHASE;
mcbspConfig.tx.frame_length1    = 0;
mcbspConfig.tx.word_length1     = WORD_LENGTH_32;
mcbspConfig.tx.frame_ignore     = NO_FRAME_IGNORE;
mcbspConfig.tx.data_delay       = DATA_DELAY0;

mcbspConfig.tx.update           = FALSE;
mcbspConfig.srg.update          = FALSE;

EXIT_ERROR(mcbsp_config(dev,&mcbspConfig));

/*************************************************************************/
/* configure CODEC                                                       */
/*************************************************************************/
EXIT_ERROR(codec_init());

codec_change_sample_rate(sampleRate, TRUE);

/* D/A 0.0 dB atten, do not mute DAC outputs                             */
EXIT_ERROR(codec_dac_control(LEFT, 0.0, FALSE));
EXIT_ERROR(codec_dac_control(RIGHT, 0.0, FALSE));

evm_led_enable(1);
evm_led_enable(0);

mcbsp_cont_async_send(dev, (unsigned char *)pBuffer,NULL,
                      numBytes,FALSE,NULL,NULL );

delay_msec(CONT_TONE_PB_SECS*1000*duration);
dev->tx_dma.continuous= FALSE;

evm_led_disable(1);
evm_led_disable(0);

/* mute DAC outputs                                                      */
EXIT_ERROR(codec_dac_control(LEFT, 0.0, TRUE));
EXIT_ERROR(codec_dac_control(RIGHT, 0.0, TRUE));

delay_msec(1000);
/* Send out reconstruction signals */
```

```
 filter(numBytes/2, pBuffer, tpBuffer, h_bytes/2, h, 0);
 for (ii=1; ii<=RenumBytes/2; ii+=2)
  tpBuffer[ii] = 0;
 filter(numBytes/2, tpBuffer, pReBuffer, h_bytes/2, h, 0);

 qmf(h, h_bytes/2);

 filter(numBytes/2, pBuffer, tpBuffer, h_bytes/2, h, 0);
 for (ii=1; ii<=RenumBytes/2; ii+=2)
  tpBuffer[ii] = 0;
 filter(numBytes/2, tpBuffer, pReBuffer, h_bytes/2, h, 1);

 EXIT_ERROR(codec_dac_control(LEFT, 0.0, FALSE));
 EXIT_ERROR(codec_dac_control(RIGHT, 0.0, FALSE));

 evm_led_enable(1);
 evm_led_enable(0);

 mcbsp_cont_async_send(dev, (unsigned char *)pReBuffer,NULL,
                       RenumBytes,FALSE,NULL,NULL );

 delay_msec(CONT_TONE_PB_SECS*1000*duration);
 dev->tx_dma.continuous= FALSE;

 evm_led_disable(1);
 evm_led_disable(0);

 EXIT_ERROR(codec_dac_control(LEFT, 0.0, TRUE));
 EXIT_ERROR(codec_dac_control(RIGHT, 0.0, TRUE));

 delay_msec(1000);


 free(pBuffer);
 return(OK);
exit_err:
 free(pBuffer);
 return(ERROR);
}


main()
{
 int status;
 Mcbsp_dev dev;

 /*************************************************************************/
 /* Initialize EVM                                                        */
 /*************************************************************************/
 if (evm_init())
 {
```

```
  printf("ERROR returned from evm_init()\n");
  return(ERROR);
 }

 led_blink( 2, 1000 );

 mcbsp_drv_init();

 dev= mcbsp_open(0);
 if (dev == NULL)
 {
  printf("Error opening MCBSP 0    \n  ");
  return(ERROR);
 }

 /*************************************************************************/
 /* Codec Tone Generator using DMA autoinitialization Example             */
 /*************************************************************************/

 status = CodecExContTonePb( dev, BOTH, 2 );
 if (status == ERROR)
 {
  printf("Error calling CodecExContTonePb()\n  ");
  return(ERROR);
 }

 mcbsp_close(dev);

 printf("Thats All Folks\n");
 led_blink( 100, 100 );

 return(OK);
}
```

VITA AUCTORIS

John Zhengluo Qiu was born in China in 1969. He obtained a Bachelor of Engineering degree from the department of Automatic Control at Southeast University, 1991. He is currently a candidate for the Master of Applied Science degree in the department of Electrical & Computer Engineering at the University of Windsor. He hopes to graduate in summer 2000.